# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE INGENIERÍA

CANCELACION ADAPTIVA DE INTERFERENCIA

**T E S I S**

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A

HUGO ARMANDO CUEVAS ATZIN

Director: M. en C. Ricardo Ruiz Boullosa

Codirector: Dr. Bohumil Psenicka

CIUDAD UNIVERSITARIA 1997.

*Un agradecimiento muy especial al Dr. Felipe Orduña Bustamantes por toda su invaluable asesoría en cuanto a la aclaración de conceptos determinantes para el entendimiento de algunos temas, así como en programación y por proporcionarme algunos programas que sirvieron para el desarrollo de este trabajo.*

**A la memoria de mis difuntos abuelos:**

**Juan Atxin Cabrera**

**y**

**José Guadalupe Cuevas González**

**Agradecimientos:**

Primeramente agradezco a mis padres: Armando Cuevas y Margarita Atzin que sin nada a cambio me han ofrecido todo su apoyo, cariño y comprensión. A ellos les debo lo que hasta hoy he logrado ser. También agradezco a mis hermanas: Tere , Juanita y Claudia que siempre han estado cerca de mi apoyándome en lo que les es posible. A Miguel A. Sánchez que incondicionalmente ha sido mi fiel amigo y compañero "cósmico". A la Universidad Nacional Autónoma de México que me ha educado y ofrecido sus servicios, instalaciones y demás. A todos los compañeros de tesis, servicio social, trabajadores e investigadores del Centro de Instrumentos de la UNAM.

El más sincero agradecimiento a la mujer que siempre ha estado junto a mi ofreciéndome su calidez y especial compañía.

Finalmente, agradezco a mi tutor y maestro: Ricardo Ruiz Boullosa por toda la atención y conocimientos que me brindó al desarrollar esta tesis.

# CONTENIDO

# Capítulo 1

## INTRODUCCIÓN

En los últimos años se ha desarrollado un área especial de la ingeniería eléctrica paralela a los grandes descubrimientos científicos y tecnológicos de disciplinas como la electrónica, las comunicaciones eléctricas, la teoría de control y la computación. Dicha área es el procesamiento de señales digitales que a la vez se divide en: análisis de sistemas y señales, análisis y desarrollo de filtros digitales, análisis espectral, identificación de sistemas, arquitectura de computadoras, etc.. Por otro lado, en teoría de control se han desarrollado sistemas de control adaptivos que en contraposición a los sistemas de control lineales e invariables en el tiempo, cambian su función de transferencia en el tiempo para adaptarse a cambios externos al sistema o para buscar su funcionamiento óptimo de acuerdo a un patrón bien definido. Estas dos disciplinas han permitido desarrollar filtros digitales adaptivos para aplicaciones en el área de la ingeniería eléctrica. Dentro de las aplicaciones de estos filtros adaptivos se encuentran: cancelador de interferencias, predicción de señales, modelación de sistemas, etc.. Para el caso del cancelador de interferencia, las señales de interés pueden ser de diversos tipos, por ejemplo, en un canal de comunicación como el teléfono, la señal de voz puede ser interferida por el eco producido por este medio y esto significa ruido en la señal de interés, así, el sistema intentará cancelar el ruido a la señal de voz; otro caso puede ser la señal de un electrocardiograma (ECG) de una madre embarazada que es recibida por un sensor al intentar leer la del feto, para este caso la señal ECG de la madre significa ruido, por lo que el sistema deberá cancelar la señal ECG de la madre.

Pues bien, el objetivo de esta tesis está dirigido al desarrollo, análisis y construcción de un filtro adaptivo digital para la cancelación de interferencias, particularmente en señales de audio. El sistema solo pretende lograr cancelar la interferencia para tener una buena inteligibilidad de las señales de interés sin preocuparse en su fidelidad.

La tesis presente se realiza para obtener el grado universitario de licenciado en Ingeniería en Computación, además, forma parte del proyecto de investigación denominado Control Adaptivo de Campos Sonoros que se lleva acabo actualmente en la sección de acústica del Centro de Instrumentos de la UNAM dirigido por el Dr. Felipe Orduña Bustamantes y el M. en C. Ricardo

Ruiz Boullosa y del cual participé como becario en la misma institución. El trabajo de investigación es meramente expositivo en el cual aplico conceptos del procesamiento de señales digitales y teoría de control adaptivo.

En el capítulo 2 se presentan algunos conceptos generales del análisis de sistemas y señales, además de proporcionar algunas herramientas matemáticas útiles para la comprensión de temas posteriores. En este capítulo se trata someramente el análisis de sistemas y señales procurando dar conceptos básicos. El capítulo 3 proporciona algunos antecedentes para el desarrollo de filtros óptimos digitales, se trata el problema de predicción lineal. En el capítulo 4 se analizan los algoritmos adaptivos y las configuraciones básicas de funcionamiento para la minimización del error del conjunto de ecuaciones normales planteadas en el problema de predicción. El algoritmo empleado en la minimización del error es el LMS que se examina igualmente en este capítulo. El capítulo 5 trata algunos conceptos de los sistemas adaptivos así como de las aplicaciones más comunes de estos. El diseño del filtro digital adaptivo se desarrolló en un procesador de señales digitales TMS320C30, por lo cual, el capítulo 6 presenta una breve descripción de los procesadores para señales digitales, fijando la atención en el TMS320C30, así como en el *software* disponible para el desarrollo de programas que funcionen en él. El capítulo 7 cubre el desarrollo del filtro y la descripción del sistema en general, es aquí donde se presentan los códigos de los programas creados en lenguaje C, también se realiza la simulación del sistema en Matlab©. Después de haber desarrollado el sistema, se realizaron una serie de pruebas para evaluar su funcionamiento, aplicando diversas señales de interferencia, es en este capítulo donde se dan a conocer los resultados.

Esta tesis está basada en los trabajos de Widrow presentados en la revista *Proceedings of the IEEE* , así como en el libro basado en el desarrollo de estos estudios, también en algunos libros especializados en filtros óptimos así como bibliografía básica del área de procesamiento de señales digitales.

Finalmente, agradezco a mi tutor M. en C. Ricardo Ruiz por toda la paciencia que mostró durante el desarrollo de esta tesis, así como al Dr. Felipe Orduña por su valiosa ayuda en la elaboración de algunos programas y asesorías. También agradezco al Ing. Antonio Pérez por el apoyo técnico que me brindó para realizar las pruebas debidas.

# Capítulo 2

## INTRODUCCIÓN A LOS SISTEMAS Y SEÑALES

El hombre desde su aparición, ha tenido la inquietud de conocer las causas que provocan los fenómenos naturales, dándoles una explicación incluso religiosa. Esta inquietud ha permitido la aparición y desarrollo de la ciencia cuya principal tarea es la de conocer todo lo que existe a su alrededor por sus principios y causas, confiando a ésta la explicación de muchos fenómenos naturales. Para esto, el hombre ha tenido la tendencia de conceptualizar los fenómenos que se presentan en nuestro medio ambiente a través de abstracciones o entes matemáticos y se han desarrollado disciplinas que con la ayuda de algunas herramientas matemáticas y de otras áreas del conocimiento, facilitan la comprensión y el estudio de dichos fenómenos. Considerando que una señal representa a una cantidad física que es función de variables como el tiempo, distancia, temperatura, presión o alguna otra variable en un fenómeno físico, el análisis de sistemas y señales se encarga del estudio de éstas. En este primer capítulo, presentaré algunos conceptos básicos de esta disciplina y de filtros digitales, además, como el fin de esta tesis es desarrollar un filtro en un procesador digital para la cancelación de interferencia, también se describirán los procesos necesarios de conversión que requieren las señales para su manipulación por el procesador digital.

### 2.1 SEÑALES Y SISTEMAS

Partiendo de la interacción básica de los seres vivos con el medio ambiente, a toda excitación proveniente del exterior corresponde así mismo una reacción, se puede utilizar esta propiedad para caracterizar todo tipo de fenómeno, sea éste físico, biológico, social, político, económico, etc. y se puede representar con un esquema como se muestra en la fig. 2.1.


Fig. 2.1 Representación de un fenómeno natural.

Los principios fundamentales de la teoria de sistemas y señales se basan en esta premisa tomando en cuenta ciertos principios matemáticos y a través de la idea fundamental de representar al fenómeno en cuestión por una "caja negra" o sistema. En este contexto, la excitación del sistema representa la señal de entrada al mismo, mientras que la reacción representa la señal de salida. Así, la caracterización de cualquier fenómeno puede ser representada con el diagrama de la fig. 2.2. El término análisis de sistemas y señales generalmente se refiere a la ciencia de analizar e interpretar las señales producidas por procesos físicos en función del tiempo, dado que es una de las variables más importantes en la vida del hombre, aunque también es común representarlos en función de su frecuencia para conocer otras características de la señal y del sistema.


Fig. 2.2 Caracterización de un fenómeno.

Las señales que se presentan en los fenómenos físicos de interés pueden clasificarse en diversos tipos, por ejemplo, en la fig. 2.3 se muestran tres tipos de señales diferentes de acuerdo a su naturaleza, la primera es una señal transitoria, éstas representan la respuesta de un sistema que inicia y cesa dentro de un determinado intervalo de tiempo; la segunda es una señal periódica, este tipo de señales se caracterizan porque se repiten en intervalos de tiempo fijos y la última es una señal aleatoria, la característica de este tipo de señales que su comportamiento es imprevisible. No importa cual sea el tipo de señal que se desee estudiar, la teoría de análisis de sistemas y señales es aplicable a todas ellas.

La clasificación de los sistemas se determina de acuerdo a las características que constituyen al mismo, también de las características de las señales con las que trabaja o una combinación de ambas. Así, se puede reconocer a los sistemas analógicos que procesan señales analógicas. Una señal analógica es aquella que es continua en un intervalo de tiempo determinado. Por otro lado, también se conocen a los sistemas digitales que en contraposición a los sistemas analógicos, son sistemas que trabajan con señales digitales. Una señal digital es una secuencia de números finita o infinita que además está cuantizada. Por ejemplo, en la fig. 2.4 se muestra el diagrama de bloques de dos sistemas, el primero es un sistema analógico cuyos componentes son dos dispositivos eléctricos: un resistor y un capacitor, que producen una salida en función de su señal de entrada y el valor de sus componentes. El segundo de los sistemas es un sistema digital cuya salida depende igualmente del valor de sus componentes: sumador, multiplicador y retardador; además del valor digital de la señala de entrada.

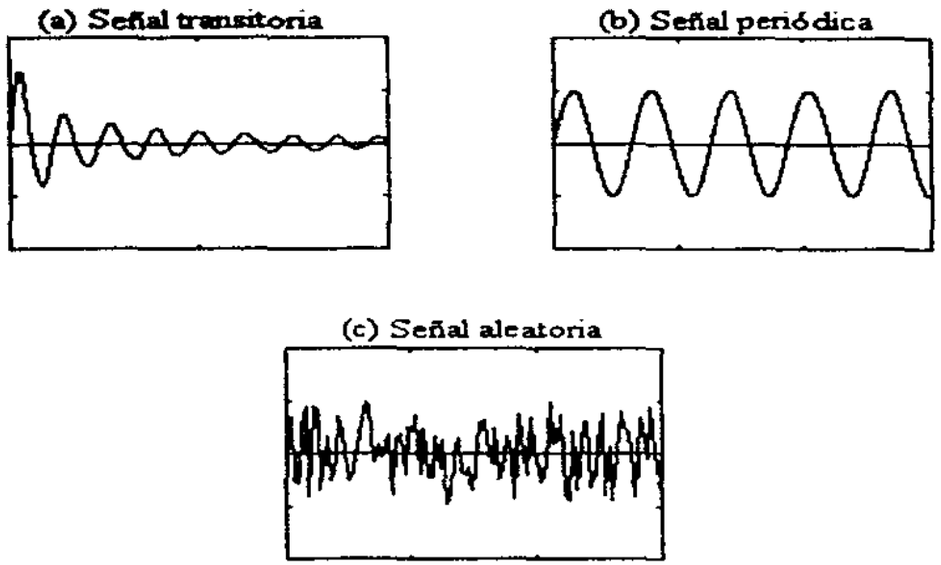



Fig. 2.3 Tipo de señales: a) Señal transitoria; b) Señal periódica; c) Señal aleatoria.

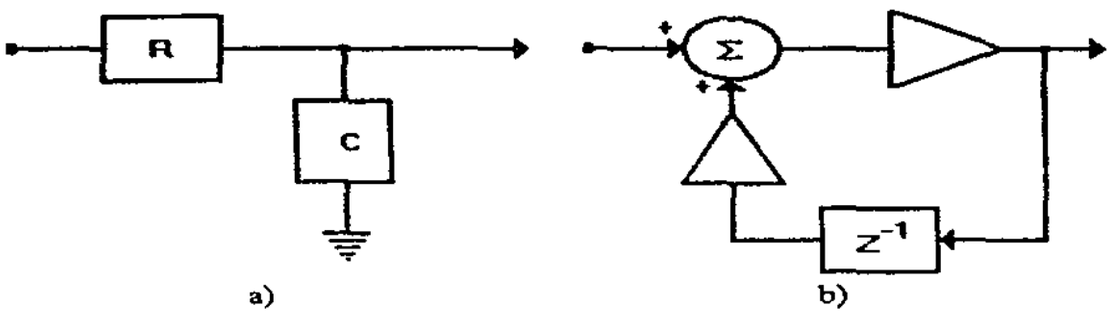



Fig. 2.4 Diagrama en bloques de un: a) Sistema analógico; b) Sistema digital.

Existen un gran número de aplicaciones donde se requiere del análisis de sistemas y señales que alcanzan muchas áreas y disciplinas como en las comunicaciones, sistemas de control, sismología, química, física y muchas más; además se han desarrollado nuevas formas de aplicación y nuevas áreas. Actualmente, gracias al desarrollo de los microprocesadores que son creados con tecnología VLSI [1], se ha incrementado el desarrollo de sistemas de procesamiento de señales digitales, cuya función es la de realizar una tarea específica en un sistema complejo. Como las señales de interés en el área de análisis de sistemas y señales se presentan en cantidades físicas como la velocidad, temperatura, presión, desplazamiento, intensidad, etc., estas son convertidas en un potencial eléctrico por medio de un transductor, para posteriormente ser

[1] Very Large Scale of Integration (Gran escala de integración).

digitalizadas a través de un convertidor analógico-digital, ADC[2]. Esta secuencia de números o secuencia de muestreo cuantizada puede ser almacenada para un procesamiento posterior o puede ser procesada en el mismo instante que es convertida, si así lo requiere la aplicación. Las operaciones más comunes del procesamiento de señales digitales son tres: la primera es referente al cálculo del espectro en frecuencia de una señal que generalmente nos muestra la distribución de la amplitud, fase, potencia o energía de la señal. La segunda de las operaciones es el filtrado digital, *vid. infra*, p. 17, cuya función es la de eliminar ciertas componentes de una señal en función de su frecuencia o reducir el ruido que puede interferir en la señal. La última de las operaciones es la correlación, que es la comparación de dos señales, puede ser la comparación de una señal por sí misma (autocorrelación) o con respecto a otra (correlación cruzada). Las señales a las cuales se aplican estas operaciones son señales digitalizadas. En la fig. 2.5, se muestra el diagrama de bloques de las operaciones típicas en el procesamiento de señales digitales.

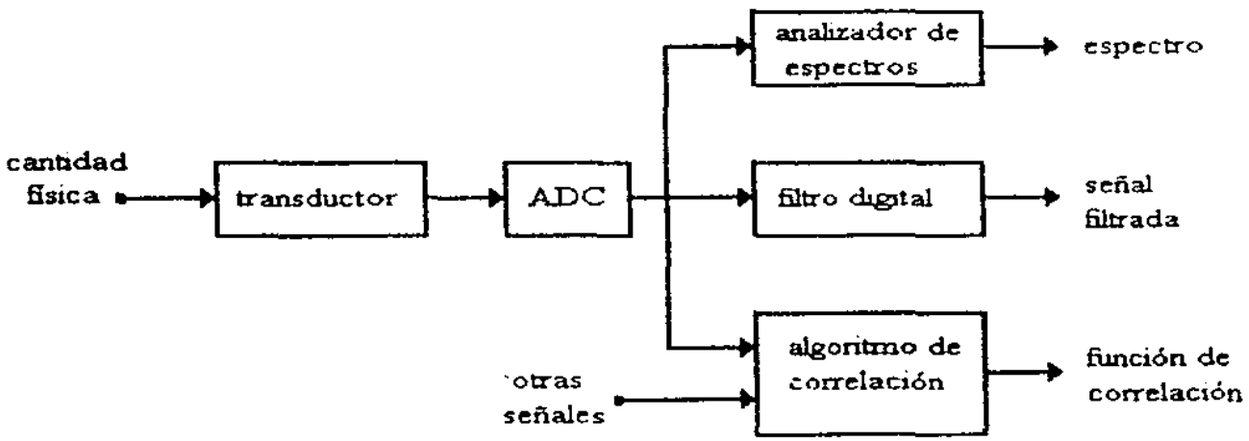



Fig. 2.5 Operaciones típicas del procesamiento de señales digitales.

Como mencioné anteriormente con el desarrollo de la tecnología VLSI, se han desarrollado un gran número de microprocesadores de bajo costo que trabajan a grandes velocidades. A este tipo de microprocesadores se les conoce con el nombre de procesadores de señales digitales, DSP[3], para más información sobre este tipo de procesadores, *vid. infra*, cap. 6; por el momento es importante comentar que debido a la gran velocidad de operación y a la versatilidad de los DSP, en la actualidad se han sustituido una gran parte de los antiguos sistemas analógicos por sistemas digitales. La mayoría de las aplicaciones de los sistemas digitales requiere de una señal analógica de salida por lo que existirá una última etapa del proceso que es la de conversión digital-analógica, DAC[4].

[2] Anolog to Digital Convertion (Conversión Analógica a Digital).
[3] Digital Signal Processor (Procesador de Señales Digitales).
[4] Digital to Analog Convertion (Conversión Digital a Analógica).

## 2.2 MUESTREO Y CUANTIZACIÓN

Al momento de realizar la conversión de las señales, se puede establecer un fundamento teórico común a los sistemas digitales y analógicos considerando a la secuencia de números procesada por el sistema digital como si fueran muestras de la señal continua que a su vez pudieran ser procesadas por el sistema analógico. Dada una señal continua f(t), se puede obtener una secuencia de muestras $f_m = f_0, f_1, f_2, \ldots$ que se deriva de la función f(t) agrupando los valores de f(t) al instante $t = 0, T, 2T, \ldots$, en una secuencia ordenada, este proceso se observa en la fig. 2.6. Al proceso de extraer el conjunto de muestras a partir de la señal original se le conoce como muestreo. El conjunto de muestras $f_m$ representa a la señal analógica que se quiere procesar y por lo tanto, se tiene que hacer énfasis en algunas de las propiedades básicas de la información que se transmite acerca de la señal original f(t).

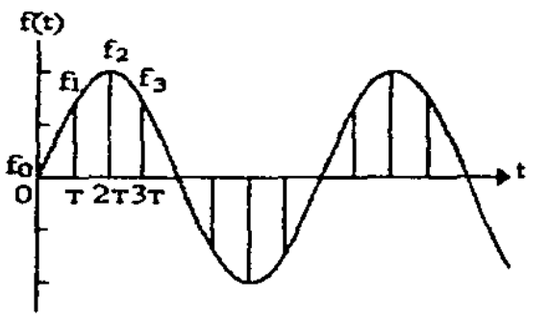



Fig. 2.6 Secuencias de muestras de una señal continua.

Supóngase que se tiene una señal f(t) que es muestreada en intervalos de tiempo constantes con una señal de impulso unitario. Una señal impulso unitario o señal *delta* se puede definir como una secuencia de muestras que vale la unidad cuando su argumento es cero y la función vale cero para cualquier otro argumento.

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \qquad (2.1)$$

En un sentido más amplio se puede definir esta señal cuando existe un desplazamiento temporal denotado por k y está dada por la expresión:

$$\delta(n-k) = \begin{cases} 1, & n = k \\ 0, & n \neq k \end{cases} \qquad (2.2)$$

la secuencia de muestras de la señal f(t), $f_m$ puede crearse multiplicando f(t) por la función de impulso unitaria $\delta(t - mT)$, este proceso se representa en la fig. 2.7.

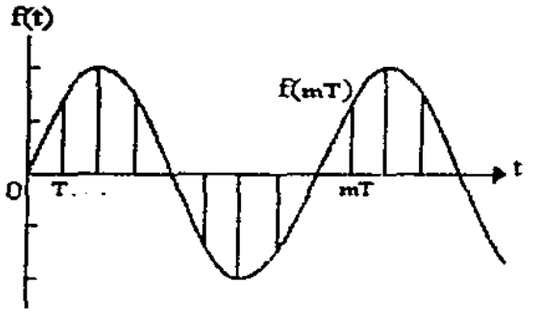



Fig. 2.7 Señal f(t) muestreada en intervalos de T segundos.

Ahora bien, ¿Cuánto debe valer esta constante para así poder reproducir la señal original a partir de la secuencia de muestreo?. T debe ser lo más pequeño posible con respecto al periodo de la señal a muestrear ya que como se observa en la fig. 2.8, es posible reconstruir funciones continuas diferentes que pasen por dos puntos de muestreo determinados, mientras más grande es el valor de T.

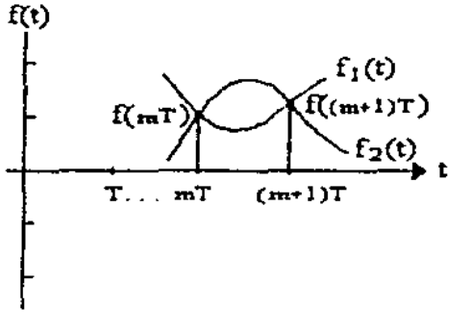



Fig. 2.8 Diferentes funciones para dos puntos de muestreo.

El teorema de muestreo impone una restricción que involucra el contenido de la frecuencia de la señal f(t) para poder reproducirla a partir de una secuencia de números $f_m$. El teorema de muestreo se enuncia de la siguiente manera: "Para poder reproducir una señal f(t) a partir de una secuencia de muestras, es necesario muestrear f(t) a una proporción mayor o igual a dos veces la frecuencia máxima de la señal f(t)"[5] Sea $\upsilon$ la frecuencia de la señal senoidal f(t). En la fig. 2.9 se muestran tres casos para valores de T diferentes. En la primera de éstas, el periodo de muestreo es exactamente igual a $1/2\upsilon$, la frecuencia de muestreo es igual a 2 veces la frecuencia de la señal f(t), de esta forma, es posible reproducir la señal sin ningún problema puesto que las muestras obtenidas forman parte de la señal f(t) aunque sus valores sean iguales a cero. Para el

[5] Stearns & Hush, *Digital Signal Analysis*, p 45.

segundo caso, el periodo de muestreo es menor a $1/2\upsilon$, esto implica que la frecuencia de muestreo es mayor a 2 veces la frecuencia de la señal f(t) obteniendo así una secuencia de muestras que sirven para reproducir f(t). En el tercer caso, el periodo de muestreo es mayor a $1/2\upsilon$ implicando que la frecuencia de muestreo es menor a 2 veces la frecuencia de la señal f(t) y se observa que se puede generar otra señal g(t), con diferente frecuencia a la señal f(t). Por lo anterior, el teorema de muestreo es fundamental cuando se decide trabajar con sistemas digitales y se requiere convertir una señal analógica-digital, ya que de esto depende la buena interpretación de las muestras que representarán a la señal analógica.

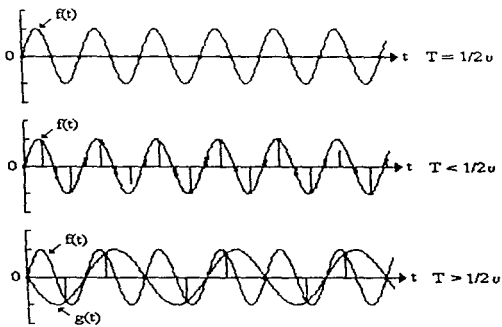



Fig. 2.9 Tres periodos de muestreo diferentes para una señal senoidal con una frecuencia $\upsilon$ a)$T = 1/2\upsilon$; b)$T < 1/2\upsilon$; c)$T > 1/2\upsilon$ para f(t).

## 2.3 CONVERSIÓN ADC Y DAC

En la práctica, cuando se trabaja con sistemas lineales e invariables en el tiempo, *vid. infra* p. 15 y con una señal continua f(t) es posible encontrar dos esquemas típicos de trabajo, presentados en la fig. 2.10. Los sistemas que trabajan con un procesador analógico reciben una señal analógica que es procesada por éste produciendo como salida igualmente una señal analógica. Estos sistemas se modelan matemáticamente por medio de ecuaciones algebraicas y ecuaciones diferenciales. Por otro lado, los sistemas digitales trabajan con un procesador digital que recibe y produce señales digitales. Un esquema típico de control usando un sistema digital se presenta en la fig. 2.11. Generalmente este tipo de controladores forman parte de sistemas complejos cuya aplicación requieren bastante precisión.

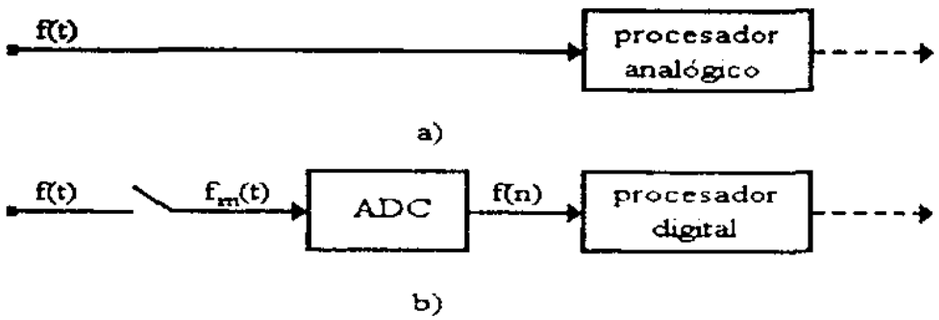



Fig. 2.10 Esquema de trabajo con señales analógicos sistema: a) analógico; b)digital.

El controlador está compuesto básicamente de un sensor cuya función es tomar una señal analógica no importando la naturaleza de ésta y convertirla a otro tipo de señal que generalmente es una señal eléctrica que igualmente es analógica; posteriormente existe una etapa de muestreo que retiene la señal eléctrica para convertirla a una secuencia de números y ser convertidos a una señal digital por un ADC. Como la señal digitalizada es procesada por un DSP, el procesador produce una señal de salida digitalizada, por lo que se requiere una última etapa de conversión DAC que realiza la conversión de la señal digital-analógica.


Fig. 2.11 Sistema típico de control digital.

## 2.3.1 CONVERSIÓN ANALÓGICA-DIGITAL

La conversión de una señal analógica-digital es un proceso necesario para acondicionar una señal analógica a un sistema digital, pero no he especificado ninguna de sus características ni como es que funciona pues no es el objetivo de esta tesis, solo aquí intentaré describir a grandes rasgos un tipo específico de ADC. En la práctica existe un gran número de circuitos electrónicos que pueden realizar esta conversión. La evaluación de un ADC gira alrededor de tres parámetros principalmente que son: precisión, velocidad y complejidad. La precisión de un ADC, depende de sus errores de linealidad en su diferenciación y/o integración de sus componentes. La velocidad estará dada en términos de número de muestras convertidas por segundo. Y finalmente su complejidad se refiere al número de componentes y dificultad de construcción tanto en sus segmentos analógicos como digitales. Estos tres parámetros interactuan entre sí de tal forma que el desempeño óptimo de cualquiera de dos parámetros afecta el buen resultado del tercero. Existen diferentes tipos de convertidores de acuerdo a su estructura y pueden estar clasificados en cualquiera de los siguientes grupos:

1. Convertidores A/D paralelo (de destello) y serial-paralelo.
2. Convertidores A/D de canalización y multiplexión.
3. Convertidores A/D seriales (aproximaciones sucesivas).
4. Convertidores A/D sobremuestreo.
5. Convertidores A/D de conteo.

La fig. 2.12 muestra la arquitectura de un circuito de conversión analógica-digital paralelo, que contiene 3 etapas. La primera de éstas es un generador de referencia, el cual debe disponer dos voltajes $V_{R+}$ y $V_{R-}$ , así como $2^n + 1$ resistencias que suministran un voltaje de referencia para cada comparador en la siguiente etapa. La segunda etapa recibe la señal a convertir por $V_{in}$ y proporciona un código de salida binaria cuyo valor es asignado en función de las salidas de los comparadores y depende del rango del voltaje de entrada, en esta etapa se distinguen un detector de sobreflujo (*overflow*) y un detector de un valor del mínimo aceptable o subflujo (*underflow*) que detectan cuando la señal de entrada a convertir se encuentra fuera de rango, es decir, del máximo o mínimo valor posibles a convertir. Finalmente, el código es convertido en una salida decimal en código binario por el circuito lógico de la tercera etapa.

Si el lector desea mayor información acerca de los ADC puede consultar [Ref. 2].

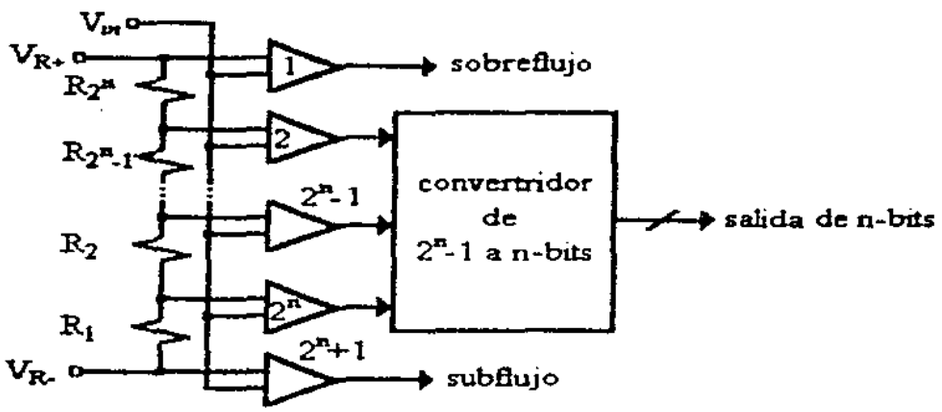



Fig. 2.12 Diagrama a bloques de un convertidor analógico-digital (ADC) tipo paralelo.

## 2.3.2 CONVERSIÓN DIGITAL-ANALÓGICA

El otro tipo de conversión al cual he hecho mención es la DAC cuya función es inversa a la del ADC, es decir, a partir de una secuencia de números codificada, se genera una cantidad de voltaje continuo. Al igual que el ADC, las arquitecturas comunes para este convertidor pueden clasificarse en la siguiente lista:

1. Convertidores D/A paralelo (de destello) y serial-paralelo.
2. Convertidores D/A canalización y multiplexión.
3. Convertidores D/A seriales (aproximaciones sucesivas).
4. Convertidores D/A sobremuestreo.
5. Convertidores D/A de conteo.

El esquema más simple de un DAC de tipo paralelo, se muestra en la fig. 2.13 que se conoce con el nombre de DAC de segmentación de corriente. Se puede observar que está compuesto por un registro binario de corrimiento de n-bits, varios *buffers* y un circuito sumador. La secuencia de números $f_m$ se va almacenando en el registro de n-bits; para decodificar cada número de la secuencia el registro debe estar sincronizado con la frecuencia de conversión. Los controladores del flujo de corriente son interruptores analógicos que se encuentran al final de cada *buffer* y que son impulsados en respuesta a los niveles lógicos de cada celda del registro, es decir, "1" ó "0". Cada generador de corriente produce una fracción de la corriente total que será sumada para así producir una señal de salida g(t) que cambia su valor solamente en los instantes de conversión. Los errores que se pueden producir en los DAC, tanto eléctricos como mecánicos se pueden minimizar usando componentes y fuentes de voltaje de precisión.

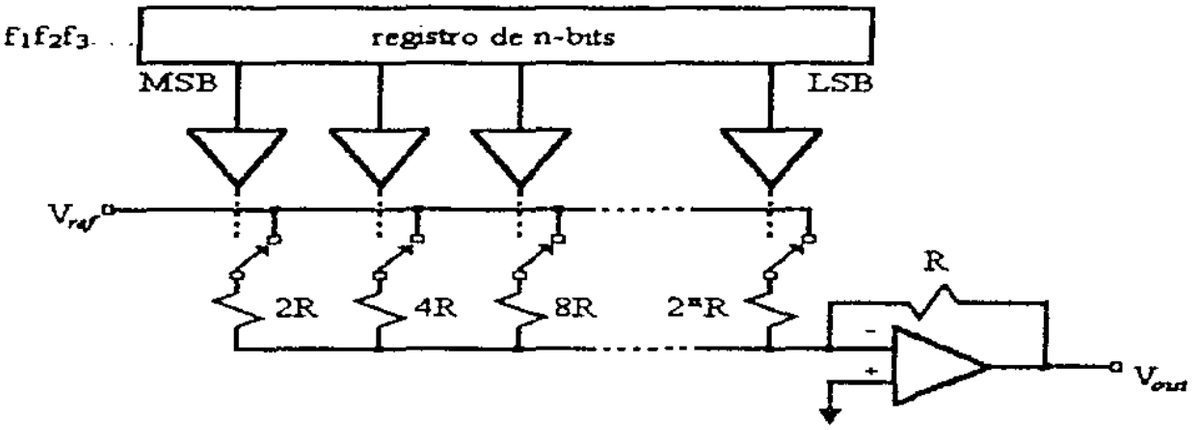



Fig. 2.13 Circuito simple de un convertidor digital-analógico (DAC) tipo paralelo de segmentación de corriente.

## 2.4. FUNDAMENTOS MATEMÁTICOS

Dejando a un lado los procesos de adecuación de las señales y retomando el tema de sistemas como interés, éstos pueden ser modelados de diversas formas, siendo más comunes los modelos en diagramas de bloques y los modelos matemáticos. El diagrama de bloques de un sistema representa esquemáticamente las funciones de cada uno de los componentes y los flujos de las señales, permitiendo la reducción de sistemas complejos a bloques más simples que representan al mismo. Sin embargo, un modelo matemático describe las características dinámicas

de un sistema y se debe tener siempre presente que la adquisición de este es la parte más importante de todo el análisis. Una vez obtenido el modelo matemático, pueden usarse diversas herramientas analíticas y computacionales con el objeto de analizarlo y sintetizarlo. Antes de continuar, es necesario fundamentar algunas herramientas matemáticas que sirven de base para el análisis de sistemas y señales. Esta tesis trata someramente dos de las herramientas matemáticas más usadas, pues el objetivo es proporcionar bases simples solo para comprensión y no de análisis. Si el lector desea mayor información, la [Ref. 6] puede ser de gran utilidad.

## 2.4.1 TRANSFORMADA DE LAPLACE

El método de la transformada de Laplace permite las resolución de ecuaciones diferenciales y la conversión de muchas funciones habituales en funciones algebraicas de una variable compleja, también es posible remplazar operaciones como la diferenciación e integración por operaciones algebraicas en el plano complejo. De esta forma se puede transformar una ecuación diferencial en una ecuación algebraica en una variable compleja y así lograr la solución de la ecuación diferencial con una tabla de transformada de Laplace. Por esta razón y debido a que los sistemas lineales e invariables en el tiempo, *vid. infra* p. 15, pueden ser modelados por ecuaciones diferenciales y algebraicas, esta transformada resulta de gran utilidad para su estudio.

La transformada de Laplace de una señal dada f(t) se obtiene de la siguiente forma:

$$\mathcal{L}[f(t)] = F(s) = \int_0^{\infty} f(t)e^{-st}dt \qquad (2.3)$$

donde: f(t) es una señal en el dominio del tiempo continuo tal que $f(t) = 0$ para $t < 0$.
s es una variable compleja $s = \sigma + j\omega$.
$\mathcal{L}$ simbolo operacional que representa la transformada de Laplace.
F(s) transformada de Laplace de f(t).

La transformada de Laplace es una modificación de la transformada de Fourier que representa a una función f(t), en el dominio del tiempo, t, en una función en el dominio de la frecuencia, ω.

Existe una transformada inversa a la transformada de Laplace, conocida como transformada inversa de Laplace, la cual permite obtener una señal f(t) a partir de la transformada de Laplace de esta señal, es decir,

$$\mathcal{L}^{-1}[F(s)] = f(t) = \frac{1}{2\pi j}\int_{a-j\infty}^{a+j\infty} F(s)e^{st}ds \qquad (2.4)$$

## 2.4.2 TRANSFORMADA-Z

La transformada-z juega un papel importante en el análisis de señales discretas y es para éstas lo que para las señales continuas es la transformada de Laplace, por lo que esta herramienta permite resolver ecuaciones en diferencias que representan un filtro digital, *vid. infra* p. 17,

cuando se trabaja con señales discretas. Por otro lado, la transformada-z resulta ser la generalización de la DFT[6], así pues, la transfromada-z de una secuencia f(n) representa a la secuencia f(n) en el dominio de la frecuencia. Para desarrollar la transformada-z se considera una señal analógica f(t) idealmente muestreada por la señal impulso unitario, *vid. supra* p. 7, de la siguiente forma:

$$f_m(t) = \sum_{k=0}^{\infty} f(t)\delta(t - kT) \tag{2.5}$$

donde: $T = 1/F_s$ es el periodo de muestreo.
$\delta(t - kT)$ es la función impulso unitario con un retardo de kT.

La transformada de Laplace de la señal muestreada $f_m(t)$ resulta ser:

$$F_m(s) = \int_0^{\infty} f_m(t)e^{-st}dt \tag{2.6}$$

$$= \int_0^{\infty} \{ f(t)\delta(t) + f(t)\delta(t - T) + \ldots \} e^{-st}dt$$

De una de las propiedades de la función impulso se tiene que:

$$\int_0^{\infty} f(t)\delta(t - kT)dt = f(kT) \tag{2.7}$$

Por lo que resulta:

$$F_m(s) = f(0) + f(T)e^{-sT} + f(2T)e^{-2sT} + \ldots = \sum_{n=0}^{\infty} f(nT)e^{-nsT} \tag{2.8}$$

Ahora, si $z = e^{sT}$, se tiene:

$$F(z) = \sum_{n=0}^{\infty} f(nT)z^{-n} \tag{2.9}$$

Se deja implícito el periodo T, por lo que f(nT) resulta f(n), consecuentemente se tiene:

$$F(z) = Z[f(n)] = \sum_{n=0}^{\infty} f(n)z^{-n} \tag{2.10}$$

esta expresión representa la transformada-z de la secuencia f(n).

También es posible obtener la secuencia f(n) a partir de la secuencia F(z) que representa la transformada-z de la secuencia f(n). A este proceso de transformación se le conoce con el nombre de transformada-z inversa y se obtiene de la siguiente manera:

$$Z^{-1}[F(z)] = f(n) = \sum_{k} \text{residuos}[F(z)z^{n-1} \text{en los k polos de } F(z)] \tag{2.11}$$

[6] Discrete Fourier Transform (Transformada de Fourier Discreta).

## 2.5 RESPUESTAS DE LOS SISTEMAS LINEALES E INVARIABLES EN EL TIEMPO DISCRETO (SLITD)

Retomando la caracterización original de que un fenómeno puede ser representado por un sistema que recibe una señal de entrada y produce una segunda señal de salida y como el fin es usar un DSP, me enfocaré a estudiar los sistemas discretos o digitales. Un sistema digital representa la correspondencia o transformación entre una secuencia x(n) y una secuencia y(n). Se usa la siguiente notación más comúnmente para la representación de la relación entre la secuencia x(n) y y(n):

$$y(n) = L\{x(n)\} \tag{2.12}$$

donde: L representa a un sistema o transformación.
y(n) salida o respuesta del sistema.
x(n) entrada al sistema.

En la fig. 2.14 se observa el esquema de la relación entre x(n) y y(n).


Fig. 2.14 Relación entrada salida de un sistema y su transformación correspondiente.

La teoría de análisis y señales se enfoca al estudio de los sistemas lineales e invariables en el tiempo (SLIT). A estos sistema se les conoce así porque cumplen con dos propiedades que son la de linealidad e invariabilidad temporal. Los sistemas lineales deben cumplir dos características que son: homogeneidad y superposición, que se describen a continuación:

Sea,

x(n), $x_1$ (n) y $x_2$ (n) señales discretas.
y(n) la señal discreta de salida del sistema.
A una constante.

**Homogeneidad:** Un sistema es homogéneo cuando:

$$y(n) = L\{Ax(n)\} = AL\{x(n)\} \tag{2.13}$$

**Superposición:** Un sistema cumple esta propiedad cuando:

$$y(n) = L\{x_1(n) + x_2(n)\} = L\{x_1(n)\} + L\{x_2(n)\} \tag{2.14}$$

Además de cumplir con estas propiedades, los SLITD deben ser invariables en el tiempo, es decir, las propiedades que rigen al sistema en un tiempo determinado no varían para cualquier otro tiempo. El modelo matemático de un SLITD está determinado por ecuaciones diferenciales

para el caso continuo, o bien, ecuaciones en diferencias para el caso discreto, por lo que las herramientas presentadas anteriormente son de gran utilidad para su análisis.

## 2.5.1 RESPUESTA A UN IMPULSO UNITARIO

Si se aplica como entrada x(n) al sistema L una señal impulso unitario δ(n), la excitación se aplica en el instante n = 0, por lo que cualquier señal de salida observada después de n = 0, debe ser característica del mismo sistema ya que la señal de entrada ha cesado. Esto se puede ver como si el sistema intentara restablecerse por sí mismo después de haber aplicado inesperadamente energía en el instante n = 0. Por esta razón se hace referencia a esta respuesta como la "respuesta natural" del sistema. Así pues, la respuesta al impulso unitario queda definida por:

$$y(n) = L\{\delta(n)\} \tag{2.15}$$

Convencionalmente se usa la siguiente notación:

$$h(n) \equiv L\{\delta(n)\} \tag{2.16}$$

Una vez estimada la respuesta al impulso unitario, h(n), la salida del sistema para cualquier entrada x(n) estará dada por:

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \tag{2.17}$$

A la ecuación anterior se le conoce como sumatoria de convolución, que se puede representar mediante el símbolo "*". Una de las características de la sumatoria de convolución es ser conmutativa, es decir, si se hace k = n - k, entonces:

$$y(n) = x(n) * h(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{2.18}$$

## 2.5.2 FUNCIÓN DE TRANSFERENCIA

También es posible representar a un sistema como una ecuación en diferencias a coeficientes constantes en la que el sistema transforma una secuencia de números de entrada x(n) en una secuencia de números de salida y(n) de la siguiente forma:

$$y(n) + a(1)y(n-1)+...+a(p)y(n-p) = b(0)x(n) + b(1)x(n-1)+...+b(q)x(n-q) \tag{2.19}$$

y como la señal de interés es y(n), entonces:

$$y(n) = -\sum_{k=1}^{p} a(k)y(n-k) + \sum_{l=0}^{q} b(l)x(n-l) \tag{2.20}$$

donde: a(k) y b(l) son los coeficientes que definen el sistema.
x(n) es la secuencia de entrada al sistema.
y(n) es la secuencia de salida del sistema.

La transformada-z de la ecuación en diferencias del sistema representa el modelo del mismo en el dominio de la frecuencia:

$$Y(z) + a(1)z^{-1}Y(z)+...+a(p)z^{-p}Y(z) = b(0)X(z) + b(1)z^{-1}X(z)+...+b(q)z^{-q}X(z) \quad (2.21)$$

o bien:

$$Y(z) = H(z)X(z) \quad (2.22)$$

donde:
$$H(z) = \frac{b(0) + b(1)z^{-1}+...+b(q-1)z^{-q+1} + b(q)z^{-q}}{1 + a(1)z^{-1}+...+a(p-1)z^{-p+1} + a(p)z^{-p}} \quad (2.23)$$

La relación que se obtiene de la transformada-z de la salida del sistema y la de la entrada al mismo se le conoce como función de transferencia y se representa mediante la siguiente relación:

$$H(z) = \frac{Y(z)}{X(z)} \quad (2.24)$$

## 2.6 FILTROS DIGITALES

En términos generales, un filtro es un sistema que toma una señal de entrada cualquiera y la convierte en otra señal. Los filtros digitales operan con señal digitales, el concepto de filtro digital resulta ser un proceso que convierte una secuencia de números llamada entrada en otra secuencia de números llamada salida. Los filtros digitales tienen una gran aplicación en todas las áreas del procesamiento de señales digitales. Estos filtros pueden ser desarrollados como una subrutina en una computadora digital (nivel programación: *software*) o como circuitos que contienen registros, multiplicadores y sumadores (nivel electrónico: *hardware*). Los filtros se pueden clasificar de diversas formas, siendo la más común aquella que caracteriza la supresión de alguna parte de la señal de entrada en términos de su frecuencia. Por lo que, se reconocen los filtros pasa-bajas, pasa-bandas, supresor de bandas, etc.. La teoría que gira alrededor de los filtros es bastante amplia y para fines de esta tesis no es necesario entender más que conceptos básicos.

De la ecuación (2.22) y (2.23), se obtiene una expresión que representará a la salida de un sistema, que en este caso representa a un filtro, en términos de z de la siguiente forma:

$$Y(z) = \frac{b(0) + b(1)z^{-1}+...+b(q-1)z^{-q+1} + b(q)z^{-q}}{1 + a(1)z^{-1}+...+a(p-1)z^{-p+1} + a(p)z^{-p}}X(z) \quad (2.25)$$

De esta ecuación, se puede observar que se tienen tres casos para la salida en función de los valores de p y q. Cuando p = 0, al filtro se le conoce como FIR[7], no recursivo, transversal, filtro exclusivamente de ceros (*all-zero*) o de media móvil (MA: *moving average*). Cuando q = 0, se obtiene un filtro conocido como IIR[8], recursivo, filtro exclusivamente de polos (*all-pole*) o autoregresivo (AR: *autoregressive*). Si tanto p como q son mayores a cero, al filtro se le puede llamar IIR, recursivo, filtro de polos y ceros (*pole-zero*) o autoregresivo de media móvil

[7] Finite Impulse Response (respuesta al impulso finita).
[8] Infinite Impulse Response (respuesta al impulso infinita).

(ARMA). Los términos AR, AM y ARMA están más relacionados con el concepto de modelado de procesos estocásticos.

En el diseño de filtros digitales, resulta útil el conocimiento de los polos y ceros de su función de transferencia. Los polos de una función de transferencia son los valores de z, ya sea reales o complejos para los cuales esta se hace infinita, mientras los ceros son simplemente los valores de z para los cuales la función de transferencia es igual a cero, es decir:

$$H(z) = \frac{\text{ceros}}{\text{polos}} \tag{2.26}$$

Estos polos y ceros pueden representarse en el plano complejo z que se muestra en la fig. 2.15.

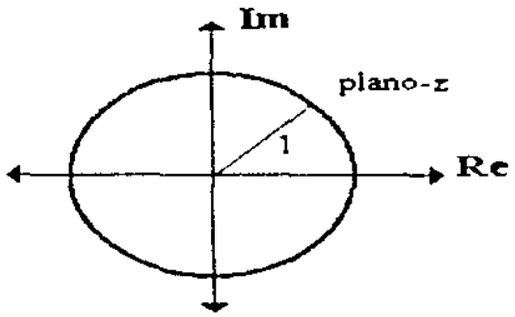



Fig. 2.15 Plano-z para la ubicación de polos y ceros de una función de transferencia.

Dada la función de transferencia H(z), la obtención de los polos se realiza igualando el denominador de H(z) a cero y se determinan las raíces de esta ecuación, para el caso de los ceros, el numerador se iguala a cero e igualmente se encuentran las raíces de la ecuación.

La secuencia de ponderación (coeficientes) de un filtro digital depende de la localización de sus polos y ceros en el plano-z. Una propiedad muy importante de esta secuencia es la convergencia o divergencia a cero, es decir, si el filtro es estable o inestable. La estabilidad de un filtro digital está relacionada con la posición de los polos de su función de transferencia de la siguiente forma: si todos los polos de esta función se localizan dentro del círculo unitario, es decir, sus magnitudes son menores a uno, entonces el filtro es estable. Si uno o mas polos se encuentran fuera de este circulo, el filtro es inestable. Por otro lado, los ceros pueden localizarse en cualquier parte del plano-z sin que esto altere la estabilidad del filtro.

## 2.6.1 FILTROS FIR

La salida de un filtro FIR que se obtiene de la ecuación (2.25), cuando p = 0, resulta:

$$Y(z) = \left(b(0) + b(1)z^{-1} + \ldots + b(q-1)z^{-q+1} + b(q)z^{-q}\right)X(z) \tag{2.27}$$

O bien,

$$Y(z) = H(z)X(z) \tag{2.28}$$

donde: $$H(z) = \sum_{l=0}^{q} b(l)z^{-l} = \frac{b(0)z^{q} + b(1)z^{q-1} + b(2)z^{q-2} + \ldots + b(q)}{z^{q}}$$

Aplicando la transformada-z inversa a la ecuación (2.27) se obtiene:

$$y(n) = \sum_{l=0}^{q} b(l)x(n-l) \tag{2.29}$$

que representa la respuesta del filtro FIR en el dominio del tiempo. En la figura 2.16 se muestra el diagrama de bloques de un filtro FIR.

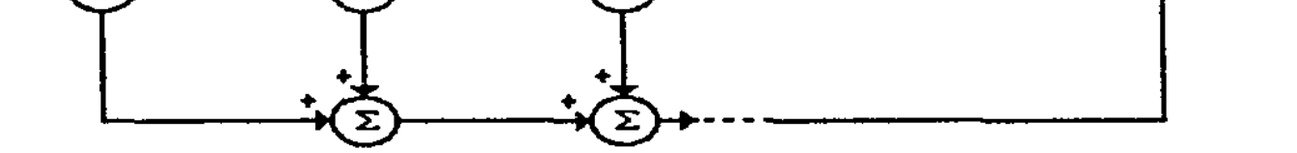



Fig. 2.16 Diagrama de bloques de un filtro FIR.

Un filtro FIR es inherentemente estable debido a que su función de transferencia contiene q polos en el origen. De la ecuación (2.29) se observa que este tipo de filtros pueden ser fácilmente desarrollados a partir del conocimiento del valor x(n) y valores pasados a él, es decir, no se requiere el conocimiento de valores pasados a la salida del mismo. Los filtros FIR pueden fácilmente diseñarse para garantizar la linealidad de su fase. En la [Ref. 5] se describe una demostración de esta característica. Sin embargo, estos filtros también tienen ciertas desventajas, la más importante es que requieren una gran cantidad de coeficientes para tener una mayor aproximación a la respuesta en frecuencia deseada.

## 2.6.2 FILTROS IIR

Si en la ecuación (2.25) el valor de q = 0, se tiene un filtro IIR que se representa con la siguiente expresión:

$$Y(z) = \frac{1}{1 + a(1)z^{-1} + \ldots + a(p-1)z^{-p+1} + a(p)z^{-p}} X(z) \tag{2.30}$$

O bien,

$$Y(z) = H(z)X(z) \tag{2.31}$$

donde: $$H(z) = \frac{z^p}{\sum_{k=1}^{p} a(k)z^{-k}} = \frac{z^p}{z^p + a(1)z^{p-1} + a(2)z^{p-2} + \ldots + a(p)}$$

En la fig. 2.17 se representa el diagrama de bloques de un filtro IIR con q = 0.

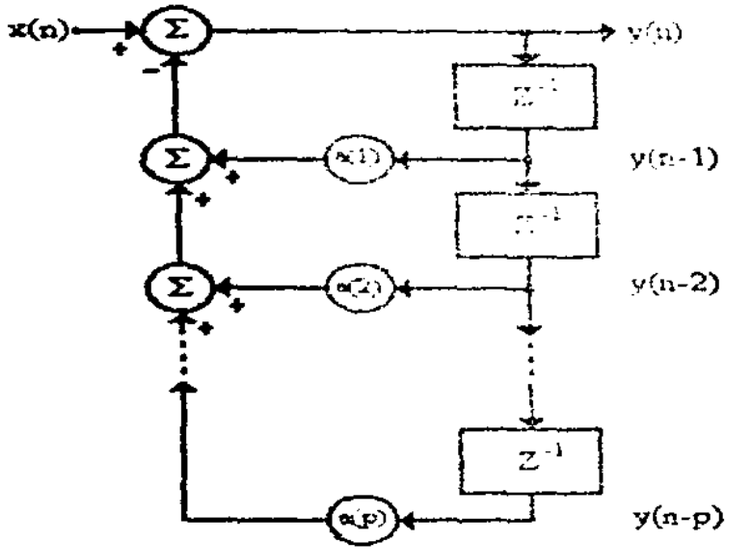



Fig. 2.17 Diagrama de bloques de un filtro IIR.

Por otro lado, si los valores de p y q son mayores a cero, entonces la expresión (2.25) resulta:

$$Y(z) = \frac{b(0) + b(1)z^{-1} + \ldots + b(q-1)z^{-q+1} + b(q)z^{-q}}{1 + a(1)z^{-1} + \ldots + a(p-1)z^{-p+1} + a(p)z^{-p}} X(z) \qquad (2.32)$$

O bien, $$Y(z) = H(z)X(z) \qquad (2.33)$$

donde: $$H(z) = \frac{\sum_{l=0}^{q} b(l)z^{-l}}{\sum_{k=1}^{p} a(k)z^{-k}} = \frac{b(0)z^q + b(1)z^{q-1} + b(2)z^{q-2} + \ldots + b(q)}{z^p + a(1)z^{p-1} + a(2)z^{p-2} + \ldots + a(p)} \qquad (2.34)$$

Existen diversas estructuras para la realización de estos filtros, en la fig. 2.18 se presenta el diagrama de bloques de un filtro IIR forma directa I con p y q mayores a cero.

La principal desventaja de un filtro IIR es que para su diseño se requiere del conocimiento de los valores pasados de la respuesta del sistema, además si p es mayor a cero, también se requiere del

conocimiento de los valores pasados de la señal de entrada x(n). En muchos casos resulta muy difícil estabilizar los filtros IIR, además éstos alteran la fase y/o la forma de la señal filtrada.

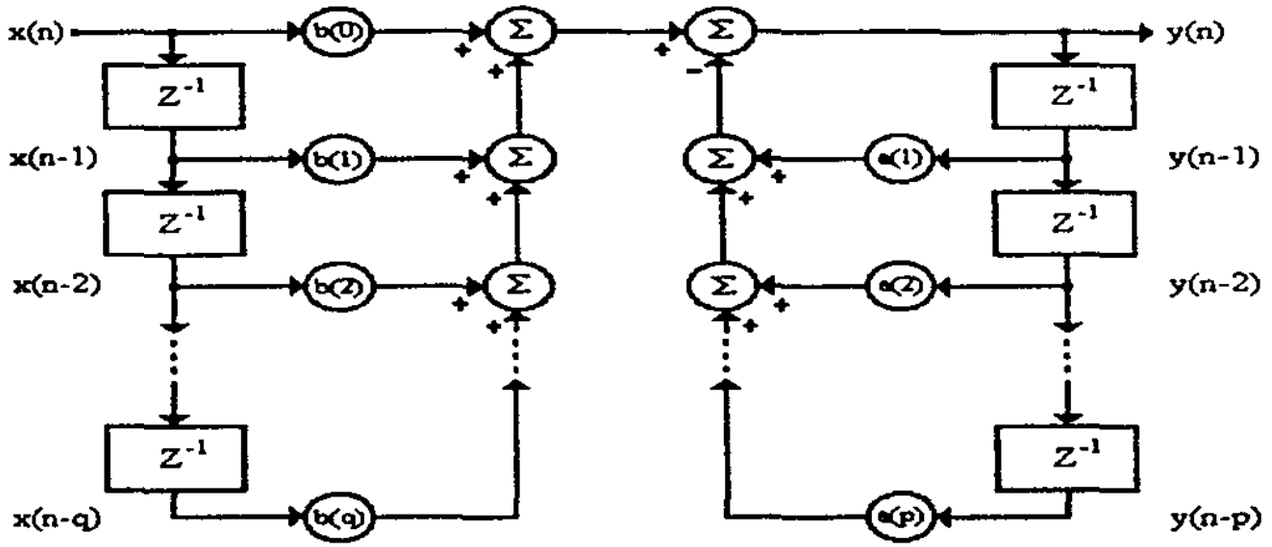



Fig. 2.18 Diagrama de bloques de un filtro IIR con estructura de forma directa I.

# Capítulo 3

## PREDICCIÓN LINEAL Y FILTROS ÓPTIMOS LINEALES

En el capítulo 2 se expusieron algunos principios que fundamentan a la teoría de sistemas y señales introduciendo conceptos básicos sobre filtros digitales. Pues bien, el sistema creado en esta tesis es un filtro digital implementado a nivel de *software*, más específicamente es un filtro digital adaptivo, es decir, sus coeficientes son variables en el tiempo y cambian bajo un criterio específico. A estos filtros se les suele nombrar como filtros óptimos. El diseño de este tipo de filtros está basado en la estimación de ciertos parámetros, que son sus coeficientes, usando algunos criterios estadísticos. Este tercer capítulo trata someramente la teoría de los filtros óptimos partiendo del proceso de predicción lineal.

### 3.1 PREDICCIÓN LINEAL

Uno de los problemas más frecuentes en el procesamiento de señales digitales es la de predicción de un valor futuro o pasado de un proceso estocástico a partir de un conjunto de valores pasados del mismo. El término de proceso estocástico o proceso aleatorio se usa "[...] para describir el desarrollo temporal en un intervalo definido de un fenómeno estadístico de acuerdo a ciertas leyes probabilísticas"[1]. El estudio de la predicción lineal antecede a los filtros óptimos que fundamentan el desarrollo del sistema de cancelación de interferencia de esta tesis.

La operación de predicción lineal recae en dos casos: (1) a partir de un conjunto de muestras $x(n-1), x(n-2), \ldots, x(n-p)$ se realiza una estimación de $x(n)$, denotando esta estimación con $\hat{x}(n)$. Esta predicción corresponde a la predicción de un paso en el futuro medido con respecto al tiempo conocida como predicción lineal de un paso en el futuro o simplemente predicción lineal futura. (2) Se usa un conjunto de muestras $x(n), x(n-1), \ldots, x(n-p+1)$ para realizar una

[1] Haykin S., *Adaptive Filter Theory*, p.78.

predicción $\hat{x}(n-p)$ de la muestra $x(n-p)$ conocida con el nombre de predicción lineal hacia el pasado. Trataré someramente por separado cada uno de los dos casos mencionados.

### 3.1.1 PREDICCIÓN LINEAL FUTURA

Comenzaré el análisis con el caso de predicción de un valor futuro de un proceso estocástico estacionario a partir de la observación de los valores pasados del proceso. "Un proceso estocástico es estacionario si sus propiedades estadísticas son invariables para un desplazamiento en el tiempo"[2] Se dice que la predicción del valor x(n) es lineal porque estará dada por la combinación lineal de los valores $x(n-1), x(n-2), \ldots, x(n-p)$ multiplicados por un peso, al conjunto de valores de peso se le conoce como coeficientes o secuencia de ponderación. La predicción lineal de x(n), denotado por $\hat{x}(n)$ resulta:

$$\hat{x}(n) = \sum_{k=1}^{p} a(k)x(n-k) \tag{3.1}$$

donde:
- a(k) representa los pesos de la combinación lineal, llamados coeficientes de predicción del predictor lineal de orden p.
- x(n-k) representa la señal de entrada al proceso en el instante n-k.

La diferencia entre el valor x(n) y la estimación de éste se le suele llamar error de predicción futura representada por f(n) y determinada por la siguiente expresión:

$$f(n) = x(n) - \hat{x}(n) = x(n) - \sum_{k=1}^{p} a(k)x(n-k) \tag{3.2}$$

En la fig. 3.1 se observa el diagrama de bloques del predictor lineal futuro de un paso usando un filtro transversal, *vid. supra* p.17.

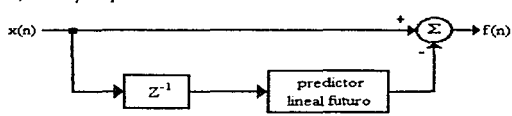



Fig. 3.1. Predictor lineal futuro de un solo paso.

### 3.1.2 PREDICCIÓN LINEAL HACIA EL PASADO

Para analizar el caso de predicción lineal pasada, se considera la secuencia de datos $x(n), x(n-1), \ldots, x(n-p+1)$ de un proceso estocástico estacionario y se desea estimar el valor de

[2] *Ibidem.*

x(n-p) de dicho proceso. Para esto, se realiza una predicción lineal pasada de orden p. Así pues, la estimación del valor de x(n-p), denotado por $\hat{x}(n-p)$ se obtiene:

$$\hat{x}(n-p) = \sum_{k=0}^{p-1} b(k)x(n-k) \tag{3.3}$$

donde: b(k) representa los pesos de la combinación lineal, llamados coeficientes de predicción del predictor lineal de orden p.
x(n-k) representa la señal de entrada al proceso en el instante n-k.

Como en el caso del predictor lineal futuro, aquí también se produce una diferencia entre el valor real x(n-p) y su estimación formulada en la siguiente ecuación:

$$g(n) = x(n-p) - \hat{x}(n-p) = x(n-p) - \sum_{k=0}^{p-1} b(k)x(n-k) \tag{3.4}$$

o bien,

$$g(n) = \sum_{k=0}^{p} b(k)x(n-k) \tag{3.5}$$

con b(p) = 1.

En la fig. 3.2 se presenta el proceso de predicción lineal hacia el pasado como se hizo para el caso anterior.

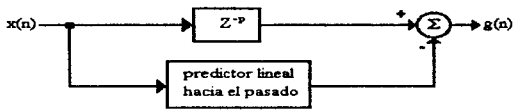



Fig. 3.2 Sistema de predicción lineal hacia el pasado de orden p.

Los dos casos de predicción vistos anteriormente producen un error que se debe minimizar de tal forma que el valor real y el valor estimado sean muy parecidos. Existe una gran número de aplicaciones que recaen sobre estos casos, por lo que se han desarrollado diversos procedimientos para minimizar el error y obtener valores lo más semejante posible. Esto ha conducido al desarrollo del filtrado lineal óptimo.

## 3.2 ESTIMACIÓN MMSE

Ahora bien, para encontrar el valor mínimo del error producido en los problemas de la sección anterior, se plantea una ecuación general para la estimación de un valor deseado d(n) a partir de una secuencia de muestras x(n) que será procesada por un filtro con respuesta al impulso unitario h(n) para producir una salida y(n) que sea lo más parecida posible a d(n). El esquema de este problema se observa en la fig. 3.3. Se puede observar que el error producido por

la diferencia entre la señal deseada d(n) y la salida del filtro y(n), está dada por la siguiente ecuación:

$$e(n) = d(n) - y(n) = d(n) - h(n) * x(n) = d(n) - \sum_k h(k)x(n-k) \tag{3.6}$$

El rango k de la sumatoria depende del tamaño y tipo de filtro con el que se implementará y por el momento no se especifica.

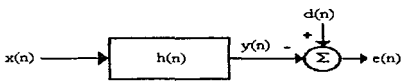



Fig. 3.3. Planteamiento general de optimización.

Existen diferentes procedimientos para minimizar la ecuación (3.6) y así obtener la mejor estimación de la señal d(n). Si el lector está interesado en conocer los diversos métodos de minimizar la ecuación (3.6) puede consultar [Ref. 3].

El procedimiento de la estimación lineal MMSE[3], resulta ser un buen método, pues se puede obtener sólo del conocimiento del primer y segundo momento de los datos. Más aún, si no están disponibles pueden estimarse directamente de estos. Además, la estructura de la estimación lineal MMSE puede fácilmente modificarse para tener la forma de un filtro FIR. Se define el error cuadrático medio como:

$$MSE = \xi = E\left[e^2(n)\right] \tag{3.7}$$

la minimización se obtiene diferenciando $\xi$ con respecto a cada coeficiente h(l) e igualando el resultado a cero.

$$\frac{\partial \xi}{\partial h(l)} = \frac{\partial E\left[e^2(n)\right]}{\partial h(l)} = 2E\left[e(n)\frac{\partial e(n)}{\partial h(l)}\right] \tag{3.8}$$

Sustituyendo (3.6) y desarrollando la derivada parcial dentro del paréntesis, resulta:

$$\frac{\partial e(n)}{\partial h(l)} = \frac{\partial}{\partial h(l)}\left\{d(n) - \sum_k h(k)x(n-k)\right\} = -x(n-l) \tag{3.9}$$

sustituyendo (3.9) en (3.8), se tiene que:

$$\frac{\partial \xi}{\partial h(l)} = 2E\left[e(n)(-x(n-l))\right] \tag{3.10}$$

sustituyendo (3.6) en (3.10), resulta:

$$= 2E\left\{\left[d(n) - \sum_k h(k)x(n-k)\right]\left[-x(n-l)\right]\right\} \tag{3.11}$$

[3] Mínimum Mean Squared Error (Mínimo Error de la Media Cuadrática).

$$= 2E\left[-x(n-l)d(n) + \sum_k h(k)x(n-k)x(n-l)\right] \tag{3.12}$$

$$= 2\left\{E\left[-x(n-l)d(n)\right] + \sum_k h(k)E\left[x(n-k)x(n-l)\right]\right\} \tag{3.13}$$

Finalmente se obtiene:

$$\frac{\partial \xi}{\partial h(l)} = 2\left[\sum_k R(n-k,n-l)h(k) - p(n-l,n)\right] \tag{3.14}$$

donde:

$$R(n-k,n-l) = E\left[x(n-k)x(n-l)\right]$$
$$p(n-l,n) = E\left[x(n-l)d(n)\right]$$

R(n-k,n-l) representa la autocorrelación de la señal de entrada x(n) y p(n-l,n) es la correlación entre la señal de entrada x(n) y la señal de salida d(n). Igualando a cero, la expresión (3.14) resulta:

$$\sum_k R(n-k,n-l)h(k) = p(n-l,n) \tag{3.15}$$

esta ecuación forma un conjunto de ecuaciones algebraicas conocidas como ecuaciones normales. La solución de la ecuación (3.15) produce el filtro óptimo. Al resultado se le conoce como solución del MMSE.

## 3.3 SOLUCIÓN DE LAS ECUACIONES NORMALES

En la sección anterior se ha planteado un conjunto de ecuaciones lineales que produce la solución del filtrado óptimo para el problema sobre estimación. Pues bien, existen diversos algoritmos para la solución de dicha ecuación y obtener así los coeficientes del filtro. Dos de los algoritmos más eficientes y comunes son el de Levison-Durbin y el de Schur. El primero fue originalmente propuesto por Levison (1947) y modificado más tarde por Durbin (1959) por lo que es conocido como algoritmo Levison-Durbin. El segundo algoritmo es atribuido a Schur (1917). El algoritmo Levison-Durbin es apropiado para un procesamiento en serie y tiene una complejidad de $O(p^2)$ operaciones[4], mientras que el de Schur también opera para $O(p^2)$ operaciones, pero cuando se usan procesadores paralelos las operaciones pueden ejecutarse en un tiempo O(p). Ambos algoritmos aprovechan la propiedad de simetría Toplitz[5] inherente en la matriz de autocorrelación de entrada. Por el momento dejo pendiente los algoritmos de optimización, me enfocaré en la solución de las ecuaciones normales.

### 3.3.1 FILTROS WIENER

Para desarrollar un filtro óptimo retomo el problema de la estimación de una señal que se desea reproducir a partir de una señal compuesta por ella misma y un ruido no deseado. Este problema se presenta a menudo en las aplicaciones del procesamiento de señales. A partir de una

[4] O(.) se usa esta notación para el análisis asintótico de un algoritmo.
[5] Una matriz es llamada Toplitz si los elementos de la diagonal principal son idénticos.

señal de entrada x(n), que consiste en la suma de una señal deseada, s(n) y una señal de ruido no deseado o interferencia, w(n), se desea diseñar un filtro que mantenga la señal deseada s(n) y elimine la señal de ruido w(n). Así, el estimador se limitará a ser un filtro lineal de respuesta al impulso h(n), cuya salida tendrá que aproximarse a una secuencia deseada específica, d(n). El esquema del problema ahora planteado se puede observar en la fig. 3.4.

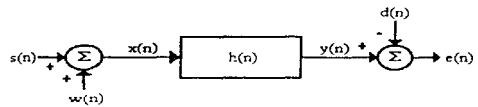



Fig. 3.4 Diagrama del filtro de estimación de una señal en presencia de un disturbio o señal de ruido.

La secuencia de entrada al filtro está dada por la suma de la secuencia s(n) y el ruido w(n), es decir, x(n) = s(n) + w(n) y la salida estará dada por la secuencia y(n). La diferencia entre la señal deseada y la salida del filtro es la secuencia de error e(n) = d(n) - y(n). En este punto no interesa la procedencia de la señal deseada d(n), sólo se considera que existe.

Antes de continuar, es importante tener algunas consideraciones presentes, primero s(n), w(n) y d(n) son señales estacionarias con media cero. El criterio que se usará para optimizar la respuesta al impulso del filtro h(n) es por MMSE, por su mejor manipulación matemática y simplicidad. También se considera que el filtro puede ser FIR o IIR, tendiendo en cuenta que si es el caso de un filtro IIR, la secuencia de entrada x(n) estará disponible para un pasado infinito. El filtro óptimo lineal, en sentido MMSE es conocido como filtro Wiener. En esta sección analizaré por separado un filtro FIR y un IIR.

### 3.3.1.1 FILTROS WIENER FIR

Comenzaré con el análisis para el caso del filtro FIR, para lo que supongo que se tiene un filtro de orden M con coeficientes h(k) $0 \le k \le M-1$. Así, la salida del filtro y(n) depende sólo de una secuencia de datos $x(n), x(n-1), \ldots, x(n-M+1)$ y de los coeficientes h(k).

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{3.16}$$

el valor medio cuadrático del error entre la señal de salida deseada d(n) y y(n) es:

$$\xi = E\left[e^2(n)\right] = E\left[\left|d(n) - \sum_{k=0}^{M-1} h(k)x(n-k)\right|^2\right] \tag{3.17}$$

así, la minimización de $\xi$ produce el conjunto de ecuaciones lineales planteado en la ecuación (3.15) y para un filtro FIR ese conjunto de ecuaciones queda determinado en la siguiente expresión:

$$\sum_{k=0}^{M-1} R(n-k, n-l)h(k) = p(n-l) \tag{3.18}$$

Por otro lado, cuando las señales de entrada son procesos aleatorios estacionarios, ocurre lo que se denomina como formulación de autocorrelación y se tiene que:

$$R(n-k, n-l) = r(l-k); \quad p(n-l) = p(l) \tag{3.19}$$

por lo que la ecuación (3.18), resulta:

$$\sum_{k=0}^{M-1} r(l-k)h(k) = p(l) \qquad \text{para } l = 0, 1, \ldots, M-1 \tag{3.20}$$

como ya mencioné anteriormente, a este conjunto de ecuaciones lineales se le conoce como ecuaciones normales y para el caso de ser el filtro óptimo se le conoce con el nombre de ecuaciones de Wiener-Hopf. Expresando las ecuaciones en forma matricial, se tiene que:

$$\mathbf{Rh} = \mathbf{p} \tag{3.21}$$

donde:
- $\mathbf{R}$ es la matriz de correlación de entrada Toeplitz de M × M.
- $\mathbf{h}$ es el vector de coeficientes del filtro.
- $\mathbf{p}$ es un vector M × 1 de correlación entre el vector de entrada y el valor deseado.

la solución para los coeficientes del filtro óptimo están dados por:

$$\mathbf{h}_{opt} = \mathbf{R}^{-1}\mathbf{p} \tag{3.22}$$

Representando la ecuación (3.17) en forma matricial, resulta:

$$\xi = E\left[d(n)^2 + \mathbf{h}^t\mathbf{x}(n-k)\mathbf{x}(n-k)\mathbf{h} - 2\mathbf{h}^t\mathbf{x}(n-k)d(n)\right] \tag{3.23}$$

$$= E\left[d(n)^2\right] + \mathbf{h}^t E\left[\mathbf{x}(n-k)\mathbf{x}(n-k)\right]\mathbf{h} - 2\mathbf{h}^t E\left[\mathbf{x}(n-k)d(n)\right] \tag{3.24}$$

$$= \sigma_d^2 + \mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p} \tag{3.25}$$

donde: $\sigma_d^2$ es la media cuadrática de d(n)

sustituyendo (3.22) en (3.25), el mínimo MSE para el filtro Wiener resulta ser:

$$MMSE = \min\xi = \sigma_d^2 + \mathbf{h}_{opt}^t\mathbf{R}\mathbf{h}_{opt} - 2\mathbf{h}_{opt}^t\mathbf{p} \tag{3.26}$$

$$= \sigma_d^2 + \mathbf{p}\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{p} - 2\mathbf{p}\mathbf{R}^{-1}\mathbf{p} \tag{3.27}$$

$$= \sigma_d^2 + \mathbf{p}\mathbf{R}^{-1}\mathbf{p} - 2\mathbf{p}\mathbf{R}^{-1}\mathbf{p} \tag{3.28}$$

$$= \sigma_d^2 - \mathbf{p}^t\mathbf{R}^{-1}\mathbf{p} \tag{3.29}$$

### 3.3.1.2 PRINCIPIO DE ORTOGONALIDAD EN LA ESTIMACIÓN LINEAL DE MEDIA CUADRÁTICA

Las ecuaciones normales para los coeficientes del filtrado óptimo de la ecuación (3.20), pueden obtenerse directamente aplicando el principio de ortogonalidad en la estimación lineal de media cuadrática. El error de la media cuadrática $\xi$ de la ecuación (3.17) es un mínimo si los coeficientes del filtro h(k) son tales que el error es ortogonal a cada punto de los datos en la estimación, es decir,

$$E[e(n)x(n-l)] = 0 \qquad \text{para } l = 0, 1, \ldots, M-1 \tag{3.30}$$

donde: $$e(n) = d(n) - \sum_{k=0}^{M-1} h(k)x(n-k) \tag{3.31}$$

En palabras se puede enunciar lo siguiente "La condición necesaria y suficiente para que la función $\xi$ obtenga su valor mínimo, es que el valor correspondiente a la estimación del error e(n) sea ortogonal a cada muestra de entrada en la estimación de la respuesta deseada en el instante n"[6]. A esto se le conoce como principio de ortogonalidad. Por otro lado, si los coeficientes del filtro satisfacen la ecuación (3.30), el resultado del MSE es un mínimo. Visto geométricamente para un filtro de dos coeficientes se tiene que:

La salida del filtro, que es la estimación de d(n),

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{3.32}$$

es un vector en el subespacio definido por los datos, x(k). El error e(n) es un vector que va de y(n) a d(n) , como se observa en la fig. 3.5.

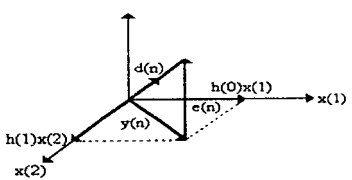



Fig. 3.5. Interpretación geométrica del principio de ortogonalidad.

Existe un corolario que puede derivarse fácilmente examinando la correlación entre la salida del filtro y(n) y el error calculado e(n).

[6] *Ibidem*, p. 163.

$$E[y(n)e(n)] = E\left[\sum_{k=0}^{M-1} h(k)x(n-k)e(n)\right] \tag{3.33}$$

$$= \sum_{k=0}^{M-1} h(k)E[x(n-k)e(n)] \tag{3.34}$$

Sea $y_{opt}(n)$ la salida producida por el filtro optimizado en MSE, con $e_{opt}(n)$ y usando el principio de ortogonalidad, se obtiene:

$$E[y_{opt}(n)e_{opt}(n)] = 0 \tag{3.35}$$

En palabras se dice: "cuando el filtro opera en condiciones óptimas, la estimación de la respuesta deseada producida a la salida del filtro $y_{opt}$ y su correspondiente estimación de error, $e_{opt}(n)$ son mutuamente ortogonales"[7].

La solución obtenida para resolver la ecuación (3.20) es única si los datos en la estimación y(n) son linealmente independientes. Para este caso, la matriz de correlación **R** es no singular. "Una matriz cuadrada **A** es no singular si $|A| \neq 0$"[8]. Por otra parte, si los datos son linealmente dependientes, el rango de **R** será menor a M, y por lo tanto, la solución no será única, en este caso, la estimación y(n) puede ser expresada como una combinación de un reducido conjunto de puntos de datos linealmente independientes igual al rango de **R**.

Debido a que el MSE es minimizado al seleccionar los coeficientes del filtro para satisfacer el principio de ortogonalidad, el residuo mínimo MSE es simplemente:

$$MMSE = E[e(n)d(n)] \tag{3.36}$$

### 3.3.1.2 FILTROS WIENER IIR

Para analizar los filtros IIR, se considera que se tiene una secuencia de muestras infinitas de los valores pasados de la señal x(n). Se plantea la salida del filtro de la siguiente forma:

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \tag{3.37}$$

el valor medio cuadrático del error entre la señal de salida deseada d(n) y y(n) es:

$$\xi_\infty = E[e(n)] = E\left[\left|d(n) - \sum_{k=0}^{\infty} h(k)x(n-k)\right|^2\right] \tag{3.38}$$

así, la minimización de $\xi_\infty$ produce el conjunto de ecuaciones lineales:

$$\sum_{k=0}^{\infty} R(l-k)h(k) = p(l) \qquad \text{para } l \geq 0 \tag{3.39}$$

[7] *Ibidem.*
[8] Deif A. S., *Advanced matrix theory for scientist and engineers*, p. 12.

calculando la transformada-z de la ecuación (3.39), se tiene:

$$R(z)H(z) = P(z) \tag{3.40}$$

donde: R(z), H(z) y P(Z) son las transformadas-z de r(n), h(n) y p(n) respectivamente.

ahora bien si H(z) representa el valor óptimo del filtro, se tiene:

$$H(z) \rightarrow H_{opt}(z) = \frac{P(z)}{R(z)} \tag{3.41}$$

La ecuación anterior no puede resolverse directamente por medio de la transformada-z, pero puede resolverse con técnicas de factorización espectral que no abarcaré aquí. El análisis para esta solución se puede encontrar en [Ref. 4].

## 3.4 LA ENERGÍA DEL MMSE

Voy a dejar a un lado los filtros IIR y continuar el desarrollo de un filtro FIR, pues este último, es el que se usa como base para el desarrollo del sistema de cancelación de interferencia. Para un valor determinado de los coeficientes del filtro FIR, se obtiene un valor de la energía del error dado por $\xi$, y esta energía del error es una medida del funcionamiento del filtro h(n). Se puede obtener este error desarrollando la expresión $\xi$, en forma matricial.

Expresando el error en forma matricial se tiene:

$$e(n) = d(n) - \mathbf{h}^t\mathbf{x}_n \tag{3.42}$$

Por lo que,

$$\xi = \sigma_d^2 + \mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p}$$

De la expresión anterior, se observa que la energía de $\xi$ es una función cuadrada de segundo orden del vector de coeficientes del filtro. Al lugar geométrico de la función de $\xi$ se le conoce con el nombre de superficie de desempeño del error o simplemente superficie de desempeño, para el filtro **h**. Para un filtro de M coeficientes, resulta ser un hiperparaboloide de dimensión M+1. Así, el mínimo valor del error se localiza en el fondo de esta superficie y ocurre cuando $\mathbf{h} \rightarrow \mathbf{h}_{opt}$. De la ecuación (3.29) y usando la expresión (3.22) se puede expresar el error mínimo en términos del filtro óptimo $\mathbf{h}_{opt}$:

$$\min\xi = \sigma_d^2 - \mathbf{p}^t\mathbf{h}_{opt} \tag{3.43}$$

El error cuadrático $\xi$, también puede expresarse en términos de una diferencia del filtro **h** y el filtro óptimo $\mathbf{h}_{opt}$. Se define un vector v conocido como vector de error

$$\mathbf{v} = \mathbf{h} - \mathbf{h}_{opt} \tag{3.44}$$

entonces, el mínimo error queda determinado por:

$$\xi = \min\xi + \mathbf{v}^t\mathbf{R}\mathbf{v} \tag{3.45}$$

Esto se puede demostrar en el siguiente desarrollo:

$$\mathbf{v}'\mathbf{R}\mathbf{v} = (\mathbf{h} - \mathbf{h}_{opt})'\mathbf{R}(\mathbf{h} - \mathbf{h}_{opt}) \tag{3.46}$$

$$= (\mathbf{h}' - \mathbf{h}'_{opt})(\mathbf{R}\mathbf{h} - \mathbf{R}\mathbf{h}_{opt}) \tag{3.47}$$

$$= \mathbf{h}'\mathbf{R}\mathbf{h} - \mathbf{h}'\mathbf{R}\mathbf{h}_{opt} - \mathbf{h}'_{opt}\mathbf{R}\mathbf{h} + \mathbf{h}'_{opt}\mathbf{R}\mathbf{h}_{opt} \tag{3.48}$$

$$= \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{h}'_{opt}\mathbf{R}\mathbf{h}_{opt} - \mathbf{h}'_{opt}\mathbf{R}\mathbf{h} - \mathbf{h}'\mathbf{R}\mathbf{h}_{opt} \tag{3.49}$$

Debido a que cada término en la ecuación anterior es un escalar y por lo tanto igual a su propia transpuesta, los dos últimos términos de la ecuación son iguales, así se obtiene:

$$\mathbf{v}'\mathbf{R}\mathbf{v} = \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{h}'_{opt}\mathbf{R}\mathbf{h}_{opt} - 2\mathbf{h}'\mathbf{R}\mathbf{h}_{opt} \tag{3.50}$$

Sustituyendo (3.50) en (3.45)

$$\xi = \min\xi + \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{h}'_{opt}\mathbf{R}\mathbf{h}_{opt} - 2\mathbf{h}'\mathbf{R}\mathbf{h}_{opt} \tag{3.51}$$

Sustituyendo (3.43) en (3.51) se obtiene:

$$\xi = \sigma_d^2 - \mathbf{p}'\mathbf{h}_{opt} + \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{h}'_{opt}\mathbf{R}\mathbf{h}_{opt} - 2\mathbf{h}'\mathbf{R}\mathbf{h}_{opt} \tag{3.52}$$

Sustituyendo (3.22), que es el vector de pesos óptimo, en (3.52)

$$\xi = \sigma_d^2 - \mathbf{p}'\mathbf{R}^{-1}\mathbf{p} + \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{p}'\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{p} - 2\mathbf{h}'\mathbf{R}\mathbf{R}^{-1}\mathbf{p} \tag{3.53}$$

$$= \sigma_d^2 - \mathbf{p}'\mathbf{R}^{-1}\mathbf{p} + \mathbf{h}'\mathbf{R}\mathbf{h} + \mathbf{p}'\mathbf{R}^{-1}\mathbf{p} - 2\mathbf{h}'\mathbf{p} \tag{3.54}$$

$$= \sigma_d^2 + \mathbf{h}'\mathbf{R}\mathbf{h} - 2\mathbf{h}'\mathbf{p} \tag{3.55}$$

Esta expresión es igual a la ecuación (3.25) y así queda demostrada la ecuación (3.45)

La expresión (3.45) representa el excedente de $\xi$ en términos de una diferencia de los coeficientes del filtro $\mathbf{h}$ y el filtro óptimo $\mathbf{h}_{opt}$ y debido a que $\mathbf{R}$ es una matriz positiva semidefinida (ver Apéndice I), entonces,

$$\mathbf{v}'\mathbf{R}\mathbf{v} \geq 0 \tag{3.56}$$

Esto demuestra que el error $\xi$ nunca es negativo para cualquier caso y que cuando $\mathbf{h}$ es $\mathbf{h}_{opt}$, entonces el error $\xi$ es mínimo, por lo que cualquier variación de $\mathbf{h}$ con respecto a $\mathbf{h}_{opt}$ sólo incrementa $\xi$. Si en la expresión (3.56) se cumple la igualdad, entonces, $\mathbf{R}$ está asociada con una matriz singular. "Una matriz cuadrada $\mathbf{A}$ es singular si $|\mathbf{A}| = 0$"[9] y permite diferentes valores mínimos. Mientras que si $\mathbf{R}$ es estrictamente positiva definida, entonces,

$$\mathbf{v}'\mathbf{R}\mathbf{v} > 0 \tag{3.57}$$

y por lo tanto, el mínimo es único.

[9] *Ibidem.*

## 3.5 FORMA NORMAL DE LA MATRIZ DE AUTOCORRELACIÓN R Y SU INTERPRETACIÓN GEOMÉTRICA

Hasta aquí he presentado el análisis para un filtrado óptimo, particularmente un filtro FIR y la energía del error medio cuadrático resultó una función conocida como superficie de desempeño del filtro h(n). Cabe mencionar que si las señales con las que se trabaja son estacionarias y tienen propiedades estadísticas invariables, entonces la superficie de desempeño permanece fija y rígida en su sistema coordenado definido por los coeficientes del filtro, mientras que si las señales no son estacionarias, pero tienen propiedades estadísticas que cambian ligeramente con el tiempo, se puede ver a la superficie como una superficie cambiante, es decir, una superficie que se mueve ligeramente en su sistema coordenado. De la expresión (3.45),se puede deducir que la orientación y la forma de la superficie de desempeño está en función de la matriz de autocorrelación **R**. Si se representa a la matriz **R** en forma normal, se pueden extraer importantes interpretaciones de la superficie de desempeño en términos de sus valores y vectores característicos que a continuación se presentan.

Los valores característicos de la matriz **R**, se pueden obtener desarrollando la siguiente ecuación homogénea:

$$[\mathbf{R} - \lambda\mathbf{I}]\mathbf{Q}_k = \mathbf{0} \tag{3.58}$$

donde: $\lambda$ es una variable escalar.
$\mathbf{Q}_k$ es un vector columna.
$\mathbf{I}$ es la matriz identidad.
$\mathbf{0}$ es un vector con todos sus elementos iguales a cero.

La ecuación anterior no tiene soluciones triviales, es decir, sus valores son diferentes de cero para $\lambda$ y $\mathbf{Q}_k$, sí y sólo sí, se cumple lo siguiente:

$$\det[\mathbf{R} - \lambda\mathbf{I}] = \mathbf{0} \tag{3.59}$$

A esta ecuación se le conoce como ecuación característica de la matriz **R**, que es una ecuación algebraica en $\lambda$ de orden M, con $\lambda_0,\lambda_1,\ldots,\lambda_{M-1}$ valores característicos. Resolviendo la ecuación (3.58) existe al menos un vector para cada valor característico $\lambda_k$, por lo que se puede representar en la siguiente expresión:

$$\mathbf{R}\mathbf{Q}_k = \lambda_k\mathbf{Q}_k \tag{3.60}$$

donde: $\mathbf{Q}_k$ es el k-ésimo vector característico asociado a $\lambda_k$.

desarrollando los vectores, se tiene:

$$\mathbf{R}[\mathbf{Q}_0\mathbf{Q}_1\cdots\mathbf{Q}_{M-1}] = [\mathbf{Q}_0\mathbf{Q}_1\cdots\mathbf{Q}_{M-1}]\begin{bmatrix}\lambda_0 & . & . & 0\\ . & \lambda_1 & . & .\\ . & . & . & .\\ 0 & . & . & \lambda_{M-1}\end{bmatrix} \tag{3.61}$$

$$\mathbf{R}\mathbf{Q} = \mathbf{Q}\Lambda \tag{3.62}$$

$$\mathbf{R} = \mathbf{Q}\Lambda\mathbf{Q}^{-1} \tag{3.63}$$

La ecuación (3.63) da la forma normal de la matriz **R**. Y debido a que **R** es una matriz simétrica, es decir, $\mathbf{R} = \mathbf{R}^t$, los vectores característicos deben ser ortogonales, es decir, $\mathbf{Q}_i^t\mathbf{Q}_j = \mathbf{0}$ para cualquier par de vectores. La matriz modal **Q** es ortonormal, por lo que se puede escribir,

$$\mathbf{Q}\mathbf{Q}^t = \mathbf{I} \tag{3.64}$$

y

$$\mathbf{Q}^{-1} = \mathbf{Q}^t \tag{3.65}$$

y por lo tanto, la matriz inversa de **Q**, siempre existe.

Finalmente, los valores característicos de la matriz de autocorrelación **R** son siempre iguales o mayores a cero, esto resulta del siguiente desarrollo:

$$\mathbf{v}^t\mathbf{R}\mathbf{v} \geq \mathbf{0} \tag{3.66}$$

debido a que **R** es una matriz positiva semidefinida (ver Apéndice I) y usando (3.63) y (3.65), se tiene:

$$\mathbf{R} = \mathbf{Q}\Lambda\mathbf{Q}^{-1} = \mathbf{Q}\Lambda\mathbf{Q}^t \tag{3.67}$$

como el vector **v** es la desviación del vector de coeficientes del filtro **h** y su valor óptimo, y si se hace **v** igual a cada columna de **Q**, así la ecuación (3.66) conserva la desigualdad para cada columna de **Q** y se puede expresar los M casos como:

$$\mathbf{Q}^t\mathbf{R}\mathbf{Q} \geq \mathbf{0} \tag{3.68}$$

sustituyendo (3.67) en (3.68), se tiene:

$$\mathbf{Q}^t\mathbf{Q}\Lambda\mathbf{Q}^t\mathbf{Q} \geq \mathbf{0} \tag{3.69}$$

y de la expresión (3.64) se tiene:

$$\Lambda \geq \mathbf{0} \tag{3.70}$$

En resumen, la representación de R en su forma normal, dice que:

1. Los vectores característicos para diferentes valores característicos de **R** son mutuamente ortogonales.
2. Los valores característicos de R son siempre reales e iguales o mayores a cero.
3. La matriz **Q** de vectores característicos pueden normalizarse (hacerse ortonormales) de manera que $\mathbf{Q}\mathbf{Q}^t = \mathbf{I}$

Estas propiedades de la matriz de autocorrelación **R** (con respecto a los vectores y valores característicos) se pueden interpretar en términos geométricos de la superficie de desempeño. De la ecuación (3.25) que describe una superficie hiperbólica en un espacio de M+1 dimensiones con sus ejes coordenados correspondientes a: $\xi, h(0), h(1), \ldots, h(M-1)$. Si se corta a esta superficie con planos paralelos a la superficie definida por el vector **h** (coeficientes del filtro), se obtiene una serie de hiperelípses concéntricas de MSE constante, y de la ecuación (3.25), la forma general de cualquiera de estas curvas proyectadas en el plano definido por **h**, es:

$$\xi = \sigma_d^2 + \mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p} \tag{3.71}$$

$$\mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p} = \xi - \sigma_d^2 \tag{3.72}$$

haciendo $\xi - \sigma_d^2 = \alpha$, se tiene:

$$\mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p} = \alpha \tag{3.73}$$

se puede trasladar de $\mathbf{h}$ a nuevas coordenadas $\mathbf{v}$, con origen en el centro de las hiperelípses. De la ecuación (3.45), $\xi = \min\xi + \mathbf{v}^t\mathbf{R}\mathbf{v}$, se tiene:

$$\mathbf{v}^t\mathbf{R}\mathbf{v} = \xi - \min\xi \tag{3.74}$$

ahora si $\xi - \min\xi = \beta$, (3.74) resulta:

$$\mathbf{v}^t\mathbf{R}\mathbf{v} = \beta \tag{3.75}$$

esta ecuación representa una elipse con centro en el origen del plano definido por $\mathbf{h}$. Con este nuevo sistema de coordenadas existen M líneas normales a las hiperelípses conocidas como ejes principales de todas las hiperelípses y por lo tanto de la superficie de desempeño, son denotadas como: $v'(0), v'(1), v'(2), \ldots, v'(M-1)$. Si se plantea una función $f(\mathbf{v}) = \mathbf{v}^t\mathbf{R}\mathbf{v}$ que define los contornos de las hiperelípses, puede obtenerse cualquier vector normal a las hiperelípses tomando el gradiente de $f(\mathbf{v})$, que a su vez también es gradiente de $\xi$, debido a que éstas difieren sólo por una constante.

$$\nabla = \left[\frac{\partial f(\mathbf{v})}{\partial v(0)}\ \frac{\partial f(\mathbf{v})}{\partial v(1)} \cdots \frac{\partial f(\mathbf{v})}{\partial v(M)}\right]^t \tag{3.76}$$

$$= 2\mathbf{R}\mathbf{v} \tag{3.77}$$

Por otro lado, cualquier vector que pase por el origen en $\mathbf{v} = 0$ debe ser de la forma $\mu\mathbf{v}$. Sólo un eje principal pasa por el origen y es normal a $f(\mathbf{v})$. Por lo tanto, se tiene,

$$2\mathbf{R}\mathbf{v}' = \mu\mathbf{v} \tag{3.78}$$

ó

$$\left[\mathbf{R} - \frac{\mu}{2}\mathbf{I}\right]\mathbf{v}' = 0 \tag{3.79}$$

donde: $\mathbf{v}'$ representa un eje principal.

Este resultado, es parecido a la ecuación homogénea definida en la ecuación (3.58), por lo que $\mathbf{v}'$ debe ser un vector característico de la matriz R. Lo anterior se puede expresar con palabras de la siguiente manera: "Los vectores característicos de la matriz de autocorrelación de entrada definen los ejes principales de la superficie de error"[10].

[10] Widrow, *Adaptive Signal Processing*, p.39.

Finalmente y a manera de resumen se tienen tres formas de representar la función $\xi$ :

$$\xi = \min\xi + \left(\mathbf{h} - \mathbf{h}_{opt}\right)^{t} \mathbf{R}\left(\mathbf{h} - \mathbf{h}_{opt}\right) \tag{3.80}$$

$$= \min\xi + \mathbf{v}^{t}\mathbf{R}\mathbf{v} \tag{3.81}$$

$$= \min\xi + \mathbf{v}'^{t}\mathbf{R}\mathbf{v}' \tag{3.82}$$

La ecuación (3.80) se encuentra en el sistema de coordenado natural, mientras que la ecuación (3.81) en el sistema coordenado trasladado y la ecuación (3.82) en el sistema coordenado principal.

Los valores característicos también tienen un importante significado geométrico que se enuncia a continuación: "Los valores característicos de la matriz de autocorrelación de entrada son las segundas derivadas de la superficie de error $\xi$ , con respecto a los ejes principales"[11]. Se puede comprobar esto fácilmente tomando (3.82) y calculando el gradiente con respecto a cada valor de $\mathbf{v}'$ . Por lo que resulta:

$$\frac{\partial\xi}{\partial\mathbf{v}'} = \left[\frac{\partial\xi}{\partial v'(0)}\frac{\partial\xi}{\partial v'(1)}\cdots\frac{\partial\xi}{\partial v'(M)}\right]^{t} \tag{3.83}$$

$$\frac{\partial^{2}\xi}{\partial\mathbf{v}'^{2}} = 2\Lambda \tag{3.84}$$

Así, los valores característicos indican la pendiente de la superficie de error sobre los ejes principales.

[11] *Ibidem*, p.40.

# Capítulo 4

## ALGORITMOS ADAPTIVOS

En el capítulo anterior se presentó el análisis para un filtrado óptimo, en el cual se planteó un conjunto de ecuaciones normales que se minimizó por MMSE; sin embargo, no he considerado los métodos existentes para lograr que los coeficientes del filtro produzcan el filtrado óptimo. Pues bien, en este capítulo expondré tres algoritmos disponibles para la minimización del MSE. Estos algoritmos son conocidos como algoritmos adaptivos y tienen la particularidad de buscar el mínimo error de la superficie de desempeño, o bien, el vector de pesos óptimo cuando solo se dispone de datos estimados sobre esta superficie, esto se debe a que en muchas aplicaciones se desconocen los parámetros del problema. Solo analizaré tres algoritmos que usan el criterio de descenso por gradiente. El interés de esta tesis es de dar una vista general de dichos algoritmos y abarcar con mayor profundidad el algoritmo LMS, ya que este algoritmo se usa para el desarrollo del sistema.

En la fig. 4.1 se muestra un esquema en el que se exponen las opciones disponibles para el diseño de algoritmos adaptivos[1]. Sólo mencionaré los algoritmos basados en el criterio de descenso por gradiente que son el método de Newton y la máxima pendiente. El algoritmo LMS se desprende del algoritmo de la máxima pendiente.

El principio general de estos métodos consiste en localizar un punto en la superficie de desempeño (un valor inicial del vector de pesos) y avanzar hacia el mínimo usando el cálculo del gradiente para indicar en que sentido se sitúa el mínimo de la superficie y una vez localizando el mínimo permanecer ahí cuando se trata de señales estacionarias e invariables en el tiempo. Por otro lado, si las señales no son estacionaria pero con propiedades estadísticas que cambian ligeramente con el tiempo, el método consiste además en rastrear al valor mínimo en tanto que la superficie se mueva en sus ejes coordenados.

---

[1] Clarkson P., *Optimal & Adaptive Signal Processing*, p. 276.

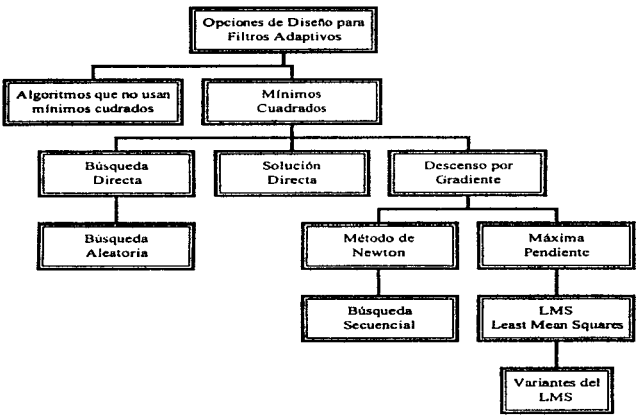



Fig. 4.1 Opciones para el diseño de filtros adaptivos.

## 4.1 IDEAS BÁSICAS DE LOS MÉTODOS DE DESCENSO POR GRADIENTE

Para introducir las ideas básicas de estos métodos, se considera el caso de un sólo peso, teniendo $r_{00}$ como único elemento de la matriz **R**, de (3.45) el vector de pesos contiene un solo elemento por lo que el error resulta:

$$\xi = \min\xi + (h - h_{opt}) r_{00} (h - h_{opt}) \tag{4.1}$$

$$= \min\xi + r_{00}(h - h_{opt})^2 \tag{4.2}$$

Como solo se tiene un solo vector, el eje h es también el eje principal y el único valor característico $\lambda = r_{00}$, por lo que se tiene:

$$\frac{\partial \xi}{\partial h} = 2\lambda(h - h_{opt}) = 2r_{00}(h - h_{opt}) \tag{4.3}$$

y su segunda derivada es:

$$\frac{\partial^2 \xi}{\partial h^2} = 2r_{00} = 2\lambda \tag{4.4}$$

Así, la segunda derivada resulta ser $2r_{00}$, en cualquier punto. En la fig. 4.2 se observa la superficie de desempeño para el caso de un solo peso.

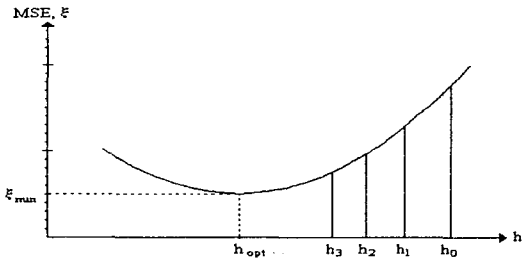



Fig. 4.2. Superficie de desempeño para un solo peso.

Como se conoce la función de desempeño, se comienza con un valor arbitrario $h_0$ y se mide la pendiente en ese punto de la curva. Se obtiene un nuevo valor $h_1$ igual al valor inicial $h_0$ mas un incremento proporcional al negativo de la pendiente. De la misma forma se obtiene un valor $h_2$ a partir de $h_1$. Repitiendo este procedimiento se puede obtener el valor óptimo $h_{opt}$. Este procedimiento se puede describir en la siguiente expresión.

$$h_{k+1} = h_k + \mu(-\nabla_k) \tag{4.5}$$

donde: k es el k-ésimo paso del proceso.
$h_k$ es el valor de h en el paso k.
$h_{k+1}$ es el valor de h en el paso k+1.

## 4.2. ALGORITMO BASADO EN EL MÉTODO DE NEWTON

Este método se deriva del algoritmo de Newton para encontrar las raíces de una función dada. El método de Newton proporciona una técnica para obtener los ceros de una función f(x) en forma iterativa. Se sabe que este método converge rápidamente para una gran clase de funciones. Nuevamente se considera el caso de una superficie de desempeño de una sola dimensión para su comprensión y posteriormente se ampliará el algoritmo para un caso multidimensional. Dada la función f(x) se desea encontrar la solución de la ecuación f(x) = 0, por lo que el método consiste en proponer un valor inicial $x_0$ y usando la primera derivada en ese

punto f'(x), se calcula la estimación $x_1$. Como se observa en la fig. 4.3, es posible encontrar el punto $x_1$, prolongando la tangente del punto $f(x_0)$ hasta que cruce al eje x.

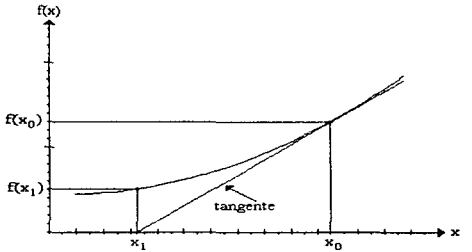



Fig. 4.3. Método de Newton para encontrar raíces de una función.

Así, de la fig. 4.3 y aplicando algunos conceptos trigonométricos, resulta que:

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \tag{4.6}$$

despejando $x_1$ de (4.6), se tiene:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{4.7}$$

y en forma general, se tiene:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{4.8}$$

la convergencia del método depende principalmente del valor inicial $x_0$ y de la naturaleza de f(x).

En la expresión (4.8) se considera que tanto la función f(x) como su derivada son conocidas. Pero para el caso en que f'(x) deba ser estimado y considerando que el valor de f(x) es conocido o fácilmente estimado, se puede estimar el valor de f'(x) usando la siguiente expresión:

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \tag{4.9}$$

Usando (4.9) en (4.8) se tiene:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \qquad \text{para } k = 0, 1, \ldots \tag{4.10}$$

Al aplicar el método de Newton en la búsqueda del mínimo de la superficie de desempeño, se comienza con una ecuación de la forma $f(x) = 0$. Por lo que $\xi'(h) = 0$ ó $\nabla = 0$, ya que se desea encontrar el mínimo de $\xi'(h)$ y así:

$$f(x) = \xi'(h) \tag{4.11}$$

por lo tanto,

$$h_{k+1} = h_k - \frac{\xi'(h_k)}{\xi''(h_k)} \qquad \text{para toda } k = 0, 1, \ldots \tag{4.12}$$

Cuando la superficie de desempeño es cuadrática, la aplicación del método de Newton permite llegar a la solución en un solo paso. Esto se puede comprobar de la siguiente forma:

Usando (4.3) y (4.4) en la ecuación (4.12) con k = 0, se tiene:

$$h_1 = h_0 - \frac{2\lambda(h_0 - h_{opt})}{2\lambda} \tag{4.13}$$

$$= h_0 - \frac{2\lambda h_0 + 2\lambda h_{opt}}{2\lambda} \tag{4.14}$$

$$= h_0 - \frac{2\lambda h_0}{2\lambda} + \frac{2\lambda h_{opt}}{2\lambda} \tag{4.15}$$

$$= h_0 - h_0 + h_{opt1} = h_{opt} \tag{4.16}$$

Por lo que se concluye que el algoritmo de Newton funciona para el caso de superficies de desempeño cuadráticas y definida para todos los valores de h. Ahora se considera el caso multidimensional, alcanzando el valor óptimo en un solo paso. El vector de pesos óptimo está dado por:

$$\mathbf{h}_{opt} = \mathbf{R}^{-1}\mathbf{p}$$

y el vector gradiente es: (ver Apéndice II)

$$\nabla = 2\mathbf{R}\mathbf{h} - 2\mathbf{p} \tag{4.17}$$

multiplicando (4.17) por $\frac{1}{2}\mathbf{R}^{-1}$ del lado izquierdo

$$\tfrac{1}{2}\mathbf{R}^{-1}\nabla = \tfrac{1}{2}\mathbf{R}^{-1}2\mathbf{R}\mathbf{h} - \tfrac{1}{2}\mathbf{R}^{-1}2\mathbf{p} \tag{4.18}$$

$$= \mathbf{h} - \mathbf{R}^{-1}\mathbf{p} \tag{4.19}$$

Usando el vector de pesos óptimo en (4.19) se obtiene:

$$\tfrac{1}{2}\mathbf{R}^{-1}\nabla = \mathbf{h} - \mathbf{h}_{opt} \tag{4.20}$$

Finalmente despejando $\mathbf{h}_{opt}$, resulta:

$$\mathbf{h}_{opt} = \mathbf{h} - \tfrac{1}{2}\mathbf{R}^{-1}\nabla \tag{4.21}$$

se puede cambiar este resultado en el algoritmo adaptivo de la siguiente forma:

$$\mathbf{h}_{k+1} = \mathbf{h}_k - \tfrac{1}{2}\mathbf{R}^{-1}\nabla_k \tag{4.22}$$

donde: $\mathbf{h}_k$ es el vector de pesos en el instante k.

$\nabla_k$ es el valor del gradiente en el instante k.

## 4.3 ALGORITMO DE LA MÁXIMA PENDIENTE

El algoritmo de la máxima pendiente también usa el gradiente para llegar al mínimo de una función dada, que en este caso se trata de la superficie de desempeño. A diferencia del algoritmo de Newton, este llega al mínimo ajustando los coeficientes del filtro en dirección del gradiente en cada iteración, los cambios no son necesariamente en dirección al mínimo de la superficie, debido a que el gradiente tiende hacia el mínimo sólo cuando el origen se encuentra en uno de los ejes principales de la superficie de desempeño. De la expresión (4.5), se puede desarrollar un algoritmo que busca el mínimo usando la pendiente en el instante k

$$\mathbf{h}_{k+1} = \mathbf{h}_k + \mu(-\nabla_k) \tag{4.23}$$

El gradiente de la superficie representado está dado por la siguiente expresión: (ver Apéndice II)

$$\nabla_k = 2(\mathbf{R}\mathbf{h}_k - \mathbf{p}) \tag{4.24}$$

y sustituyendo (4.24) en (4.23), resulta:

$$\mathbf{h}_{k+1} = \mathbf{h}_k + \mu\left[-2(\mathbf{R}\mathbf{h}_k - \mathbf{p})\right] \tag{4.25}$$

$$= \mathbf{h}_k + 2\mu(\mathbf{p} - \mathbf{R}\mathbf{h}_k) \tag{4.26}$$

La expresión anterior es el algoritmo conocido como de la máxima pendiente. En la fig. (4.4), se presenta el diagrama de flujo para los dos algoritmos expuestos hasta este momento.

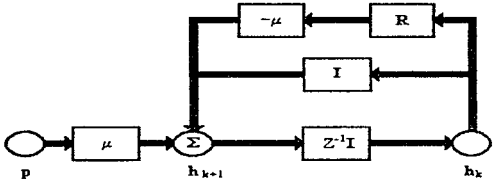



**Fig. 4.4 a) Diagrama de flujo del algoritmo basado en la búsqueda por gradiente: Máxima pendiente.**

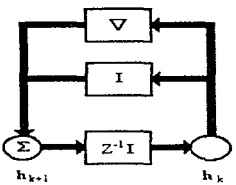



Fig. 4.4 b) Diagrama de flujo del algoritmo basado en la búsqueda por la pendiente: Método de Newton.

La estabilidad de un algoritmo basado en la búsqueda por la pendiente, depende del valor elegido de $\mu$ en la expresión (4.5). Así cuando $\mu = -\frac{1}{2}$, usado por el algoritmo de Newton se asegura la convergencia del mismo y además se logra el óptimo valor de los coeficientes en un sólo paso. Por otro lado, la condición necesaria para que el algoritmo de la máxima pendiente después de un análisis no presentado aquí, resulta ser:

$$0 < \mu < \frac{1}{\lambda_{max}} \tag{4.27}$$

donde: $\lambda_{max}$ es el máximo valor característico de la matriz **R**.

Hasta ahora se considera que se conoce el gradiente de la superficie de desempeño en cada instante k del algoritmo y además que es posible conocer la matriz **R** y el vector **p** de la superficie, pero en la mayoría de las aplicaciones y me refiero a las aplicaciones cotidianas, no es posible tener esta medición, por lo que se usa una estimación basada en un muestreo estadístico. Tal estimación es considerado como el gradiente real más un ruido adicional. No abordaré sobre el tema de la estimación del gradiente de la superficie de desempeño. Si se desea saber sobre este tema el lector podrá consultar [Ref. 9].

## 4.4. ALGORITMO LMS

El algoritmo LMS es el más simple para el ajuste de los pesos en un proceso adaptivo lineal. El desarrollo que presentaré de este algoritmo es un filtro FIR. Recordando que el error producido por la diferencia entre d(n) y y(n) está dado por:

$$e(n) = d(n) - h(n) * x(n) \tag{4.28}$$

para desarrollar un algoritmo adaptivo usando los métodos previos, se tendría que estimar el gradiente de $\xi$ tomando diferencias entre promedios pequeños de $e^2(n)$. En cambio, para construir el algoritmo LMS, se toma $e^2(n)$ por si mismo como una estimación de $\xi_k$. Por lo tanto, en cada iteración en el proceso adaptivo se tiene una estimación del gradiente de la forma:

$$\hat{\nabla}_k = \begin{bmatrix} \frac{\partial e_k^2(n)}{\partial h(0)} \\ \cdot \\ \cdot \\ \frac{\partial e_k^2(n)}{\partial h(M-1)} \end{bmatrix} = 2e_k(n) \begin{bmatrix} \frac{\partial e_k(n)}{\partial h(0)} \\ \cdot \\ \cdot \\ \frac{\partial e_k(n)}{\partial h(M-1)} \end{bmatrix} = -2e_k(n)\mathbf{x}_k \qquad (4.29)$$

Con esta simple estimación del gradiente, se puede desarrollar un algoritmo adaptivo del tipo de la máxima pendiente. Así de (4.5) $\mathbf{h}_{k+1} = \mathbf{h}_k - \mu\hat{\nabla}_k$, se tiene:

$$\mathbf{h}_{k+1} = \mathbf{h}_k + 2\mu e_k(n)\mathbf{x}_k \qquad (4.30)$$

Debido a que los coeficientes cambian en cada iteración y están basados en estimaciones imperfectas del gradiente, se espera que el proceso adaptivo sea ruidoso. Pero a pesar de esta deficiencia el algoritmo en la práctica es muy sencillo y fácil de implementar porque no requiere elevaciones cuadráticas, diferenciaciones, medición de funciones de correlación o inversión de matrices. En la fig. 4.5 se puede ver el diagrama de flujo del algoritmo LMS.

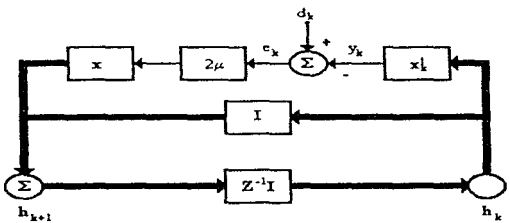



Fig. 4.5. Diagrama de flujo del algoritmo LMS.

## 4.4.1 CONVERGENCIA DEL VECTOR DE PESOS

Al desarrollar un algoritmo es importante considerar, entre otras cosas, su convergencia. En particular, lo importante de un algoritmo adaptivo, es conocer su convergencia para obtener la solución de los coeficientes óptimos, en donde se minimiza $E[e^2(n)]$. Para examinar la convergencia del vector de pesos del LMS, primero se debe hacer notar que la estimación del

gradiente no es sesgada cuando el vector de pesos se mantiene constante. "Un estimador $\hat{\theta}$ de un parámetro $\theta$, no es sesgado si $E[\hat{\theta}] = \theta$"[2]. El valor esperado de (4.29) con $\mathbf{h}_k$ igual a $\mathbf{h}$ es:

$$E[\hat{\nabla}_k] = -2E[e(n)\mathbf{x}] \tag{4.31}$$

$$= -2E[d_k\mathbf{x}_k - \mathbf{x}_k\mathbf{x}_k^t\mathbf{h}] \tag{4.32}$$

$$= 2(\mathbf{Rh} - \mathbf{p}) = \nabla \tag{4.33}$$

Debido a que el valor medio de $\hat{\nabla}_k$ es igual al gradiente real $\nabla$ (*cfr.* ecuación (4.33) con Apéndice II), $\hat{\nabla}_k$ es un estimador no sesgado.

De (4.23) se observa que $\mathbf{h}_k$ es función solo de los valores pasados del vector de entrada $\mathbf{x}_{k-1}$, $\mathbf{x}_{k-2}$, ..., $\mathbf{x}_0$. Si se asume que los vectores de entrada sucesivos son independientes en el tiempo, $\mathbf{h}_k$ es independiente de $\mathbf{x}_k$. Para procesos de entrada estacionarios y bajo esta condición, el valor esperado del vector de pesos $E[\mathbf{h}_k]$ después de un número suficiente de iteraciones converge a la solución optima de Wiener, esto es $\mathbf{h}_{opt} = \mathbf{R}^{-1}\mathbf{p}$.

$$E[\mathbf{h}_{k+1}] = E[\mathbf{h}_k] + 2\mu E[e_k\mathbf{x}_k] \tag{4.34}$$

$$= E[\mathbf{h}_k] + 2\mu\left(E[d_k\mathbf{x}_k] - E[\mathbf{x}_k\mathbf{x}_k^t\mathbf{h}_k]\right) \tag{4.35}$$

Como $\mathbf{x}_k$ y $\mathbf{h}_k$ son independientes, se tiene:

$$E[\mathbf{h}_{k+1}] = E[\mathbf{h}_k] + 2\mu(\mathbf{p} - \mathbf{R}E[\mathbf{h}_k]) \tag{4.36}$$

$$= E[\mathbf{h}_k] + 2\mu\mathbf{p} - 2\mu\mathbf{R}E[\mathbf{h}_k] \tag{4.37}$$

$$= [1 - 2\mu\mathbf{R}]E[\mathbf{h}_k] + 2\mu\mathbf{p} \tag{4.38}$$

De $\mathbf{h}_{opt} = \mathbf{R}^{-1}\mathbf{p}$, se tiene, $\mathbf{p} = \mathbf{R}\mathbf{h}_{opt}$, por lo que (4.38) resulta:

$$E[\mathbf{h}_{k+1}] = [1 - 2\mu\mathbf{R}]E[\mathbf{h}_k] + 2\mu\mathbf{R}\mathbf{h}_{opt} \tag{4.39}$$

Cambiando al sistema coordenado de ejes principales:

$$E[\mathbf{v}'_k] = (\mathbf{I} - 2\mu\Lambda)^k\mathbf{v}'_0 \tag{4.40}$$

donde:
- $\mathbf{v}'$ es el vector de pesos $\mathbf{h}$ en el sistema de ejes principales.
- $\Lambda$ es la matriz diagonal de los valores característicos de $\mathbf{R}$.
- $\mathbf{v}'_0$ es el vector de pesos inicial en el sistema de ejes principales.

De esta forma, al incrementar k sin limite, se observa que el vector de pesos en (4.40) alcanza la solución óptima sólo si el lado derecho de la ecuación converge a cero. Se mencionó anteriormente que dicha convergencia se garantiza si,

$$\frac{1}{\lambda_{max}} > \mu > 0 \tag{4.41}$$

[2] Mendenhall W. & Sincich T., *Statics for the engineering & computer sciences*, p.281.

$\lambda_{max}$ es el valor mas grande de los valores característicos, esto es el elemento mas grande de la diagonal en $\Lambda$. Así, de la ecuación (4.41), se tiene limites en $\mu$ para la convergencia de la media del vector de pesos al vector de pesos óptimo. Dentro de estos limites, la velocidad de adaptación y también el ruido en la solución de los pesos son determinados por el tamaño de $\mu$. También se nota que $\lambda_{max}$ no puede ser mayor al trazo de **R**, que es la suma de los elementos de la diagonal de **R**, esto es:

$$\begin{aligned}\lambda_{max} &\le \mathrm{tr}[\Lambda] = \sum(\text{elementos de la diagonal } \Lambda) \\ &= \sum(\text{elementos de la diagonal } \mathbf{R})\end{aligned} \tag{4.42}$$

Para un filtro transversal adaptivo, $\mathbf{R} = E[\mathbf{x}_k\mathbf{x}_k^t]$, el $\mathrm{tr}[\mathbf{R}]$ resulta: $(M)E[\mathbf{x}_k^2]$ ó M veces la potencia de la señal de entrada. De esta forma la convergencia de la media del vector de pesos se asegura por:

En general: $$0 < \mu < \frac{1}{\mathrm{tr}[\mathbf{R}]} \tag{4.43}$$

Para un filtro transversal: $$0 < \mu < \frac{1}{(M)(\text{potencia de la señal})} \tag{4.44}$$

Este es un limite más restringido para $\mu$ que en (4.41), pero es mucho más fácil de aplicar porque los elementos de **R** y la potencia de la señal generalmente pueden ser más fácil de estimar que los valores característicos de **R**.

# Capítulo 5

## SISTEMAS ADAPTIVOS

La adaptación es una característica que poseemos los seres vivos y nos es útil para evolucionar y defendernos de los cambios que nuestro medio ambiente nos provoca. La adaptación no sólo se presenta a un nivel fisiológico sino que también se puede observar en fenómenos sociales, políticos, económicos, etc.. El hombre siempre ha tomado algunos modelos de la naturaleza para desarrollar su tecnología intentando modelar ciertos fenómenos naturales para su propio beneficio y no es la excepción para el caso de la adaptación. Recientemente se ha desarrollo un campo de investigación que estudia los sistemas adaptivos y como resultado ha logrado una gran variedad de autómatas cuyas características en forma limitada se asemejan a las de un sistema viviente y a los procesos adaptivos biológicos.

En este capítulo, se introduce el concepto de sistemas adaptivos así como sus propiedades y características. También se mencionan y describen las aplicaciones más comunes de estos sistemas en forma muy superficial. Antes de comenzar, es conveniente dar una definición de un sistema adaptivo: "Un sistema de control adaptivo es un sistema que continua y automáticamente mide las características dinámicas (como la función de transferencia) de la planta, las compara con las características deseadas y usa la diferencia para variar parámetros ajustables del sistema o para generar una señal de accionamiento de modo que se pueda mantener el funcionamiento óptimo con independencia de las variaciones ambientales"[1].

### 5.1 SISTEMAS DE CONTROL ADAPTIVO

En el capítulo 2, se mencionó el concepto general de sistema, ahora bien, para que un sistema pueda clasificarse como adaptivo, debe poseer algunas características que los distinga de los sistemas lineales e invariables en el tiempo, *vid. supra* p. 15. A continuación enunciaré algunas de ellas: los sistemas adaptivos pueden adaptarse automáticamente en presencia de un

[1] Ogata K., *Ingeniería de Control Moderna*, p.855.

medio ambiente alterado; pueden ser programados para funcionar a un filtrado específico y en tomas de decisión, esto se logra con un proceso de entrenamiento. Esta cualidad permite que los sistemas adaptivos no requieran los procedimientos complicados de síntesis que generalmente se usan para los sistemas no adaptivos; deben ser capaces de extrapolar un modelo de comportamiento para ocuparse de nuevas situaciones después de haber sido entrenados en un número finito y generalmente pequeño de señales de entrenamiento o patrones; hasta cierto límite, deben repararse por ellos mismos, esto es, pueden adaptarse en torno a ciertos tipos de defectos internos.

Generalmente los sistemas adaptivos son más complejos y difíciles que los sistemas no adaptivos, pero ofrecen la posibilidad de incrementar substancialmente su funcionamiento cuando las características de la señal de entrada son desconocidas o variantes en el tiempo.

La base del control adaptivo descansa en la premisa de que existe alguna condición de operación o funcionamiento del sistema que es mejor que cualquier otro; en control adaptivo el comportamiento óptimo está definido en términos del índice de su funcionamiento, que debe elegirse al establecer los fines que éste persigue, por lo que, el índice de comportamiento debe ser confiable o ser una medida uniforme del sistema además de ser selectivo, o sea, que debe involucrar un valor óptimo claramente definido en función de los parámetros del sistema. No debe haber óptimos locales porque el sistema no estaría realizando su objetivo principal. El índice de comportamiento debe ser fácilmente aplicable y medible a sistemas prácticos.

## 5.2 PROPIEDADES GENERALES DE LOS SISTEMAS ADAPTIVOS

La propiedad principal y esencial de los sistemas adaptivos es que son variables en el tiempo y de ajuste automático. Esta característica los distingue de los sistemas lineales e invariables en el tiempo. Al desarrollar un sistema adaptivo se debe considerar todas las condiciones de entrada posibles al sistema, o al menos estadísticamente y definir bien lo que se desea que el sistema realice bajo cada una de estas condiciones. Para desarrollar un sistema adaptivo se debe elegir el sistema que más se asemeje al criterio de funcionamiento seleccionado, que generalmente es tomado *a priori* de los sistemas lineales. Cuando se diseña un sistema adaptivo, el rango completo de condiciones de entrada al sistema no puede ser conocido exactamente, aún estadísticamente, o las condiciones de desempeño cambian en el tiempo. En tales circunstancias, un sistema adaptivo que busca continuamente el grado óptimo usando un proceso de búsqueda ordenado, daría un funcionamiento superior comparado a un sistema de diseño no adaptivo.

La característica de funcionamiento de un sistema adaptivo depende entre otras cosas de sus señales de entrada, es decir, si una señal de entrada $x_1$ se aplica al sistema, entonces un sistema adaptivo se adaptará a $x_1$ y producirá una salida $y_1$. Si otra señal $x_2$ se aplica al sistema, el sistema se adaptará a $x_2$ y producirá una segunda salida $y_2$. Generalmente, la respuesta de los sistemas adaptivos será diferente para dos entradas diferentes. El principio de superposición no funciona para los sistemas adaptivos como en el caso de los sistemas lineales, es decir, si se aplica al sistema la suma de las dos entradas $x_1 + x_2$, el sistema se adaptará a esta nueva señal y producirá una salida diferente a $y_1 + y_2$. En la fig. 5.1, se presenta el diagrama del principio de no superposición de los sistemas adaptivos.

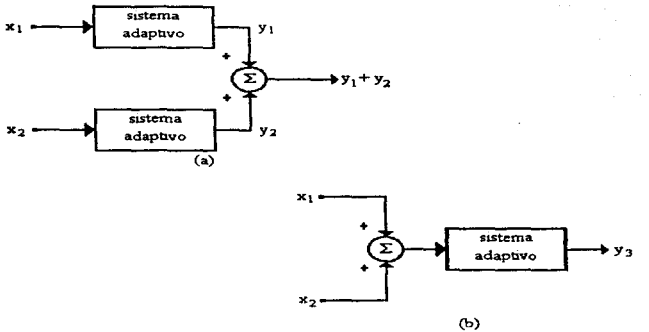



Fig. 5.1. Diagrama de bloques para el principio de no superposición de los sistemas adaptivos. (a) El sistema adaptivo produce una respuesta para la entrada $x_1$ y $x_2$ respectivamente y la suma de estas respuestas resulta ser $y_1 + y_2$. (b) El sistema adaptivo produce una salida $y_3$ para la entrada $x_1 + x_2$. Estas dos salidas $(y_1+y_2)$ y $y_3$ son diferentes.

Existen dos características en los sistemas adaptivos para distinguirlos de los sistemas no lineales y son: (1) los sistemas adaptivos son ajustables y su ajuste depende generalmente de las características promedio de tiempo finito de la señal en vez de los valores instantáneos de la señal o estados internos del sistema. (2) los ajustes de los sistemas adaptivos se realizan intencionalmente a fin de optimizar las medidas especificadas de funcionamiento. Algunos sistemas adaptivos se convierten en sistemas lineales cuando sus ajustes se mantienen constantes después de una adaptación. A estos se les conoce como: "sistemas adaptivos lineales". Estos sistemas son muy útiles, tienden a ser matemáticamente tratables y generalmente son más fáciles de diseñar que otros.

## 5.3 ADAPTACIÓN DE MALLA ABIERTA Y MALLA CERRADA

En teoría de control de sistemas en general, se hace una distinción importante entre dos clases de sistemas: sistemas de malla abierta y de malla cerrada. Pues bien, para los sistemas adaptivos se hace la misma distinción.

El proceso adaptivo de malla abierta implica hacer mediciones de las características de entrada o del medio ambiente y aplica esta información a una fórmula o a un algoritmo computacional, y usar los resultados para establecer los ajustes del sistema adaptivo. La adaptación de malla

cerrada, implica la experimentación automática con estos ajustes y el conocimiento de sus resultados con el fin de optimizar su funcionamiento. A este último proceso se le llama adaptación por "retroalimentación".

En la fig. 5.2. se observa el diagrama de bloques de los sistemas de malla abierta y el de malla cerrada respectivamente. El procesador, que previamente fue ajustado por un algoritmo adaptivo toma la señal de entrada y la procesa para obtener su salida, mientras que para el sistema de malla cerrada, el procesador es ajustado a partir de las señal de entrada y salida que sirven de información para calcular el funcionamiento óptimo del sistema y así el algoritmo adaptivo realiza el ajuste adecuado. Otros datos, en estas figuras pueden referirse al medio ambiente del sistema adaptivo, o en el caso de malla cerrada, puede ser una versión deseada de la señal de salida.

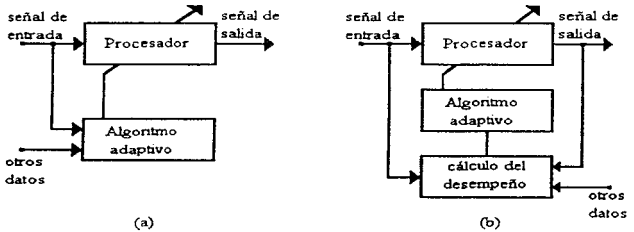



Fig. 5.2 a) Sistema adaptivo de malla abierta; b)Sistema adaptivo de malla cerrada.

La adaptación de malla cerrada puede ser fácilmente aplicable cuando el procedimiento de síntesis analítico no existe o no se conoce, es el caso donde se usan los criterios de error en lugar de usar el cálculo de la media de los cuadrados, también donde los sistemas no son lineales ó variables en el tiempo ó las señales no son estacionarias. La adaptación de malla cerrada puede usarse con efectividad en situaciones en donde los valores de los componentes de los sistemas físicos son variables o no se conocen con exactitud, así, la adaptación de malla cerrada encontrará la mejor opción de los valores ajustando sus componentes.

En caso de una falla parcial del sistema, un mecanismo de adaptación que continuamente monitorea su funcionamiento, intentará optimizarse ajustando las partes intactas; por lo que la seguridad del sistema puede ser mejorada por el uso de un proceso de retroalimentación.

La adaptación de malla cerrada no siempre es la mejor opción, en ciertas situaciones, las funciones de trabajo no tienen un único valor óptimo, entonces la automatización resulta ser un proceso incierto en tales circunstancias. En otras ocasiones, el proceso de adaptación de malla cerrada, puede ser inestable, provocando que el proceso de adaptación pueda divergir en vez de converger.

A pesar de estas posibles anomalías que se puedan presentar en un sistema adaptivo de malla cerrada, la retroalimentación es una técnica poderosa ampliamente aplicable para la implantación de un sistema de adaptación.

## 5.4 APLICACIONES DE LOS SISTEMAS ADAPTIVOS

Toda la teoría desarrollada en los capítulos precedentes tiene como finalidad desarrollar una aplicación específica de los sistemas adaptivos. Pero es importante mencionar que existen diversas aplicaciones de los sistemas adaptivos en el área del procesamiento de señales digitales. En esta sección solo abarcaré someramente cuatro de ellas, si el lector está interesado en conocer con mas detalle estas aplicaciones, puede consultar [Ref. 9]. Las aplicaciones que presentaré son: predictor adaptivo, modelado adaptivo ó identificación de sistema adaptivo, modelado inverso adaptivo (ecualización adaptiva) y cancelador de interferencia.

### 5.4.1 PREDICTOR ADAPTIVO

El caso de predicción se analizó en el capítulo 3. Esta aplicación tiene mayor uso en las comunicaciones. En la fig. 5.3, se tiene la estructura de predicción adaptiva futura, es decir, a partir de una secuencia s(n-1), s(n-2), ..., s(n-p), el procesador producirá un valor y(n) que intentará parecerse al valor actual s(n) y lograr que e(n) sea mínimo.

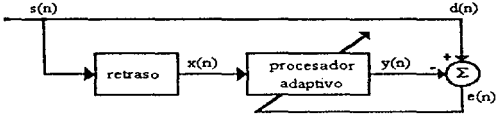



Fig. 5.3 Predictor adaptivo.

### 5.4.2 MODELADO ADAPTIVO

Estos sistemas tienen gran aplicación en control, comunicaciones y procesamiento de señales, pero también se aplica en estudios de sistemas sociales, económicos ó biológicos. Un sistema de modelado adaptivo, imita el comportamiento de un sistema físico dinámico que se puede ver como una desconocida "caja negra" el cual recibe una ó más señales del exterior para producir igualmente una ó más señales de salida. En la fig. 5.4, se tiene el diagrama de bloques de un sistema dinámico de modelado adaptivo de una sola entrada con una sola salida. En esta estructura se tiene una planta ó sistema desconocido del cual se desea modelar su comportamiento y considerando que se puede disponer de las señales de entrada y salida de la misma, es posible aplicar la misma señal de entrada a un procesador adaptivo que intentará simular la respuesta de la planta en cuestión para minimizar el error e(n). Así, la función de transferencia del procesador adaptivo una vez que ha llegado a su etapa de adaptación, será

esencialmente la misma que la función de transferencia de la planta y se dice que la planta ha sido identificada. La estructura y los valores del sistema adaptivo pueden o no ser semejantes a los del sistema desconocido, pero sus funciones de transferencia serán las mismas.

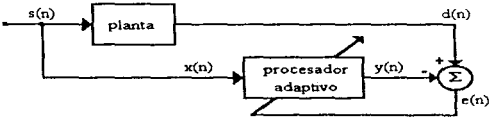



Fig. 5.4 Identificador de sistema.

## 5.4.3 MODELADO INVERSO ADAPTIVO

EL modelado inverso se aplica en el procesamiento de voz, en sistemas de comunicación y síntesis de filtros digitales, se aplica también en sistemas de control en el cual el modelado inverso de una planta desconocida se usa para generar señales de control a la planta, es igualmente aplicable para la deconvolución y ecualización. En la fig. 5.5 se muestra el esquema del modelado inverso adaptivo, en este esquema se tiene una señal s(n) que es la entrada a la planta, ésta produce una salida a la cual se añade una señal w(n) que representa un ruido. Esta señal x(n) es recibida por el procesador adaptivo que intentará parecerse lo mejor posible a la entrada s(n) de la planta. Una vez que se ha logrado la adaptación en el sistema, la función de transferencia del procesador adaptivo será el recíproco de la función de transferencia de la planta. La planta generalmente es un sistema causal, por lo que se requiere un retardo de la señal s(n), en el esquema de modelado inverso adaptivo, proporcional al tiempo que la señal s(n) utiliza al pasar por la planta.

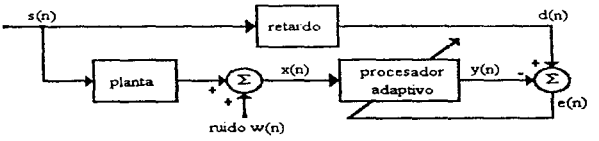



Fig. 5.5 Modelado inverso.

## 5.4.4 CANCELADOR DE INTERFERENCIA

Finalmente se presenta el cancelador de interferencia que es una de las aplicaciones más comunes en el procesamiento de señales digitales y el objetivo del trabajo de esta tesis, *vid. infra* cap. 7. El cancelador de interferencia considera que se puede recuperar una señal s(n), que ha

sido contaminada con algún tipo de interferencia w(n), y también que se dispone de una versión correlacionada con la señal de ruido w'(n). La señal w'(n) sirve de entrada al procesador adaptivo, el cual produce una señal y(n) que intenta parecerse lo más posible a w(n) para que así, e(n), sea lo más parecida a s(n). El diagrama del cancelador de interferencia se observa en la fig. 5.6.

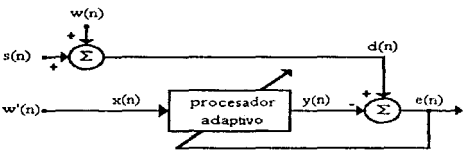



Fig. 5.6 Cancelador de interferencia.

Ahora bien, si se usa el criterio de minimización MMSE, *vid. supra* p. 25 para la implementación en estas aplicaciones, es importante mencionar que el MMSE depende del número de coeficientes de ajuste usados en la implementación del procesador adaptivo.

# Capítulo 6

## PROCESADOR DE SEÑALES DIGITALES TMS320C30

Debido al desarrollo en la electrónica digital y a la necesidad de realizar grandes cantidades de operaciones para los sistemas digitales, se han construido un gran número de procesadores que trabajan a grandes velocidades de operación, pudiendo así resolver algunos problemas en el desarrollo de sistemas que requieren un procesamiento digital de señales. A estos procesadores se les conoce como procesadores digitales de señales DSP (*Digital Signal Processing*) y se caracterizan por realizar una gran cantidad de operaciones en tiempos muy cortos, por lo que son capaces de trabajar en sistemas en tiempo real. Este capítulo da una breve historia de los procesadores de señales digitales así como las características generales del procesador TMS320C30 y del *software* requerido para el control de la tarjeta del sistema que contiene al DSP para el desarrollo del sistema de cancelación de interferencia.

### 6.1 BREVE HISTORIA DE LOS PROCESADORES DE SEÑALES DIGITALES

Los DSP comenzaron a aparecer aproximadamente a finales de la década de los 70, con el Bell Labs DPS1 y el NEC 7720. El procesador Intel 2920 apareció en 1979 e incluía convertidores ADC y DAC , para información de los convertidores *vid. supra* p. 9, en el mismo circuito integrado, pero carecía de un multiplicador por *hardware*. A principios de la década de los 70 aparecieron un gran número de microcontroladores, así como circuitos integrados que realizaban funciones de filtrado, módem, síntesis de voz y reconocimiento de voz, pero no fueron clasificados como DSP por su limitación de programación.

En 1982, Texas Instruments introdujo el TMS320C10, el primer miembro de la familia que llegó a ser la más popular entre los DSP. Su popularidad se debió al gran desarrollo de *software* y *hardware* creado alrededor de este DSP. Por aquella fecha, Hitachi introdujo el primer DSP

construido con tecnología CMOS[1], el HD61810 (HSP), que también fue el primer DSP en usar un formato de punto flotante (con una mantisa de 12 bits y un exponente de 4 bits). Un año después, Fujitsu incrementa grandemente la comercialización de los DSP programables con el MB8764 que tardaba un tiempo de multiplicación y acumulación de 120 ns.

En 1984, apareció en el mercado el Bell Labs DSP32 y llegó a ser el mejor en dos aspectos: fue el primer DSP construido por AT&T y el primer DSP de 32 bits de punto flotante. No mucho tiempo después, NEC introdujo su procesador de punto flotante de 32 bits μPD77230 con un tiempo de multiplicación y acumulación de 150 ns. El primer procesador de punto flotante en usar el formato *standard* de IEEE[2] es el Fujitsu MB86232, que apareció en 1987. Motorola no sacó ningún DSP al mercado hasta 1987 con el DSP56001, que es un DSP de punto fijo de 24 bits.

## 6.2 LA FAMILIA TMS320

La línea del producto TMS320 de Texas Instruments ofrece una familia de procesadores digitales, diseñados para soportar un amplio rango de aplicaciones en el procesamiento de señales. digitales. Estos microprocesadores/microcomputadores de 16/32 bits combinan la flexibilidad de un controlador a alta velocidad con la capacidad numérica de un procesador.

La familia TMS320 contiene el primer microprocesador MOS[3] capaz de ejecutar cinco millones de instrucciones por segundo (MIPS). Este alto desempeño es el resultado de el conjunto de instrucciones comprensivo, eficiente y de fácil programación, además de su arquitectura de "canalización". Se han incorporado instrucciones especiales tales como la multiplicación con acumulación que hacen más rápido la ejecución de algoritmos en el procesamiento de señales digitales. Se incluye un conjunto completo de instrucciones booleanas para el manejo de bits. Las capacidades de extracción de bits e interrupciones también forman parte de las características de los procesadores de la familia TMS320.

Referente a su arquitectura, esta familia utiliza la arquitectura modificada de Harvard en cuanto a velocidad y flexibilidad. En una estricta arquitectura Harvard, el programa y las memorias de datos se encuentran en dos espacios separados, permitiendo una coincidencia cabal en el ciclo de instrucción, búsqueda y ejecución. La modificación en la arquitectura Harvard de la familia TMS320 permite la transferencia entre espacios de programa y datos, incrementando la flexibilidad del dispositivo. Esta modificación en la arquitectura elimina la necesidad de una memoria ROM[4] de coeficiente separado y también maximiza la potencia de procesamiento manteniendo dos estructuras de *bus* separadas (programa y datos) para una ejecución de velocidad total.

En 1983, Texas Instruments introdujo la primera generación NMOS[5] de DSP de punto flotante, el TMS320C10, cuyas características mas sobresalientes son las de ser un procesador con multiplicador interno y trabaja con un tiempo de ciclo de instrucción de 200 ns.

[1] CMOS (Complementary Metal Oxide Semiconductor) : Semiconductor de óxido de metal complementario.
[2] IEEE (Institue of Electrical and Electronic Enginners) : Instituto de ingenieros eléctricos y electrónicos.
[3] MOS (Metal Oxide Semiconductor) : Semiconductor de óxido de metal.
[4] ROM (Read Only Memory) : Memoria de sólo lectura.
[5] NMOS (N-type Metal Oxide Semiconductor) : Semicondcutor de óxido de metal tipo N.

La segunda generación CMOS la representó el TMS320C25. Los beneficios agregados a este procesador son que incluye mayor memoria interna y trabaja con un tiempo de ciclo de instrucción de 100 ns.

La tercera generación la conforma el TMS320C30 que es un procesador de punto flotante de 32 bits, con más memoria interna y un tiempo de ciclo de instrucción de 60 ns, lo que significa 16.6 MIPS. Este procesador tiene la característica de trabajar con instrucciones en paralelo, produciendo un tiempo de ciclo de instrucción de 30 ns ó 33 MIPS. Debido a su rapidez de trabajo, se pueden realizar un gran número de instrucciones para facilitar las operaciones más comunes en el procesamiento de señales tales como el filtrado y análisis espectral.

El TMS320C40 representa la cuarta generación de esta familia. Este procesador maneja un formato de 40 bits para punto flotante y 32 bits para tipo entero, funciona a 40 y 50 MHz, cuenta con canales de comunicación paralela de 20 Mbytes con un tamaño de 6 bytes. Posee un coprocesador DMA[6] de seis canales para el CPU y operación de entrada/salida además 2 kpalabras de memoria SRAM[7] interna y un cargador de arranque en memoria ROM.

La introducción del TMS320C50 marcó el inicio de la quinta generación. Este microprocesador funciona con un ciclo de reloj 57 MHz lo que significa 28.5 MIPS con 16 bits punto fijo, contiene 10 kpalabras de SRAM interna y dos puertos seriales.

## 6.3 MICROPROCESADOR TMS320C30

El microprocesador TMS320C30 forma parte de la tercera generación de la familia TMS320 de DSP implementados en tecnología 1μm CMOS, capaces de manejar operaciones en punto flotante de 32 bits, de tipo entero y lógicas. Tiene una memoria interna de 2k palabras (32 bits cada palabra), con un *bus* de direcciones de 24 bits, lo que resulta un total de $2^{24}$ = 16.7 Mpalabras direccionables que contienen programas, datos y espacios de entrada y salida. El TMS320C30 cuenta con un ciclo de instrucción de 60 ns y la mayoría de instrucciones trabajan en sólo un ciclo. De esta forma, puede ejecutar 16.66 MIPS. Además, debido a que muchas de sus instrucciones se pueden implementar en paralelo. El TMS320C30 efectivamente puede ejecutar 33.3 MIPS. Su arquitectura permite cuatro niveles de canalización. Se fabrican en un encapsulado de tipo PGA[8] de 181 patillas. Funciona con un reloj externo de 66 MHz.

A continuación se presenta un resumen de las características más sobresalientes del TMS320C30:

- Tiempo de ejecución por instrucción de ciclos simple 60 ns que resulta un total de 16.7 MIPS.
- Memoria ROM de acceso *dual* en el circuito integrado de 4k × 32 bits.
- Dos bloques de memoria RAM[9] de acceso *dual* de 1k × 32 bits.
- *Cache* de instrucciones de 64 × 32 bits.

[6] DMA (Direct Memory Access) : Memoria de acceso directo.
[7] SRAM (Static Random Acces Memory ) : Memoria de acceso aleatorio estática.
[8] PGA (Pin Grid Array) : Arreglo en malla de las patillas.
[9] RAM (Random Access Memory) : Memoria de acceso aleatorio.

- Palabras de instrucciones y datos de 32 bits, direcciones de 24 bits.
- Multiplicador entero, ALU y registro de cilindro.
- Ocho acumuladores de precisión extendida.
- Controlador DMA en el circuito integrado para entrada/salida simultánea.
- Instrucciones de dos y tres operandos.
- Dos puertos seriales para soportar transferencias de 8, 16 y 32 bits.
- Dos *timers* de 32 bits.
- Encapsulado PGA de 181 patillas.

## 6.3.1. ARQUITECTURA DEL TMS320C30

La arquitectura del TMS320C30 responde a las demandas de los sistemas que están basados en algoritmos aritméticos sofisticados y que ofrece tanto soluciones de *hardware* como *software*. Se logra un alto desempeño en el amplio rango dinámico de las unidades de punto flotante, en su memoria interna amplia, en su alto grado de paralelismo y en su controlador de memoria de acceso directo. En la figura 6.1 se muestra el diagrama de bloques de la arquitectura del TMS320C30.

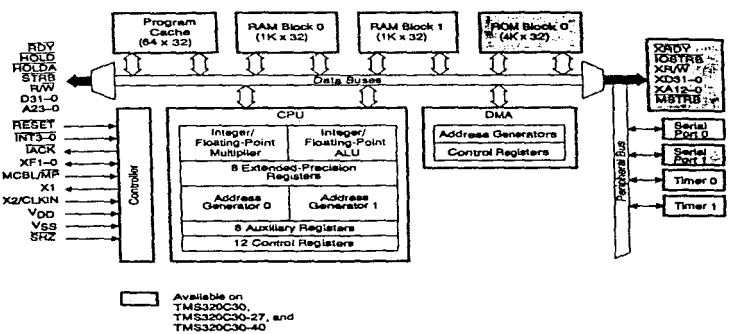



Fig. 6.1 Diagrama de bloques funcional del TMS320C30

### 6.3.1.1 UNIDAD CENTRAL DE PROCESO (CPU)

El TMS320C30 tiene una arquitectura de CPU basada en registros. El CPU tiene los siguientes componentes:

- Multiplicador de números enteros y de punto flotante.
- Unidad Aritmética Lógica (ALU) para ejecutar operaciones aritméticas de punto flotante, de enteros y operaciones lógicas.
- Registro de corrimiento de 32 bits.
- *Buses* internos (CPU1/CPU2 y REG1/REG2).
- Unidades Aritméticas de Registros Auxiliares (ARAU).
- Archivo de registros del CPU.

La fig. 6.2 muestra los varios componentes del CPU.

**MULTIPLICADOR:** El multiplicador realiza multiplicaciones de un sólo ciclo con valores de enteros de 24 bits y de punto flotante de 32 bits. Al realizar las multiplicaciones con números de punto flotante de 32 bits, el resultado es un número de punto flotante de 40 bits. Cuando el multiplicador toma valores enteros de 24 bits el resultado será un número de 32 bits.

**UNIDAD LÓGICA ARITMÉTICA (ALU):** La ALU realiza operaciones de un sólo ciclo para datos enteros de 32 bits, lógicos de 32 bits y de punto flotante de 40 bits, incluyendo las conversiones de entero y de punto flotante de un sólo ciclo. Los resultados de la ALU mantienen el formato de 32 bits para los enteros y de 40 bits para punto flotante.

**REGISTRO DE CORRIMIENTO DE 32 BITS:** El registro de corrimiento se usa para realizar corrimientos hasta de 32 bits tanto a la izquierda como a la derecha en un sólo ciclo.

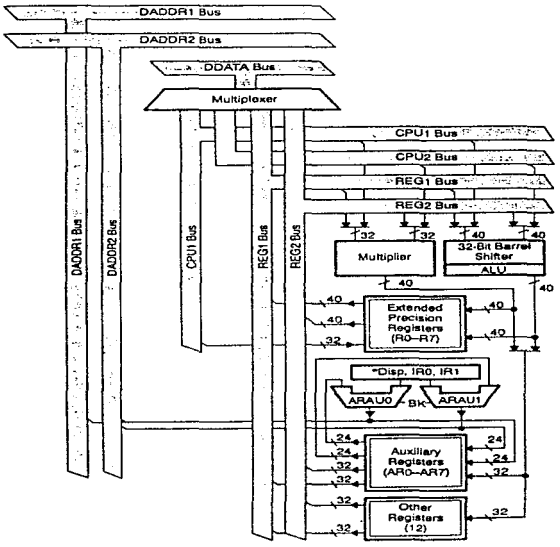
**BUSES INTERNOS CPU1/CPU2 Y REG1/REG2:** Cargan dos operandos de la memoria y dos operandos del archivo de registro permitiendo multiplicaciones paralelas y sumas/restas sobre cuatro operandos de tipo entero o punto flotante en un sólo ciclo.

**UNIDADES ARITMÉTICAS DE REGISTROS AUXILIARES (ARAU):** Las ARAU0 y ARAU1 pueden generar dos direcciones en un sólo ciclo. Las ARAU operan en paralelo con el multiplicador y la ALU. Pueden trabajar con direccionamiento de desplazamientos, registro de índice (IR0 e IR1), direccionamiento circular y de inversión de bit.

**ARCHIVO DE REGISTRO DEL CPU:** El TMS320C30 provee 28 registros en un archivo de registro multipuerto que está fuertemente acoplado al CPU. Todos estos registros pueden ser operados tanto por el multiplicador como por la ALU y pueden ser usados como registros de propósito general,; sin embargo, los registros tienen algunas funciones especiales. Los nombres de los registros y las funciones asignadas se enuncian a continuación:

- **Registros de precisión extendida (R0-R7):** Son registros de 40 bits capaces de almacenar y realizar operaciones con números enteros de 32 bits y de punto flotante de 40 bits. Para el caso de instrucciones que asumen que los operandos son números de punto flotante se usan los 40 bits, mientras que para los operandos de tipo entero sean con signo o sin él, se usan sólo los 32 bits menos significativos y los demás bits permanecen sin cambios.

- **Registros auxiliares (AR0-AR7):** Los registros auxiliares de 32 bits se pueden accesar por el CPU y ser modificados por las dos ARAU. La principal función de éstos registros es la de generar las direcciones de los 24 bits. También pueden usarse como contadores de ciclo o como registros de propósito general que pueden ser modificados por el multiplicador y por la ALU.




* Disp = an 8-bit integer displacement carried in a program control instruction

Fig. 6.2 Unidad Central de Proceso (CPU) del TMS320C30.

- **Apuntador a página de datos (DP):** El DP es un registro de 32 bits, en donde los 8 bits menos significativos del DP se usan por el modo de direccionamiento directo como un puntero a la página de datos que será direccionada. Las páginas de datos son de 64 kpalabras de largo, con un total de 256 páginas.
- **Los registros de índice (IR0, IR1):** Estos registros contienen el valor usado por las ARAU para calcular una dirección indexada.
- **Registro de tamaño de bloque (BK):** La ARAU usa el BK de 32 bits en un direccionamiento circular para especificar el tamaño del bloque de datos.
- **Apuntador al stack del sistema (SP):** El SP es un registro de 32 bits que contiene la dirección de la localidad más alta del *stack* del sistema. Una operación *push* produce un preincremento del puntero del *stack*, mientras que una *pop* produce un postdecremento en el mismo puntero. El SP es manipulado por interrupciones llamadas retornos de subrutinas y por las operaciones *push* y *pop*.
- **Registro de status (ST):** El ST contiene información relevante al estado del CPU. Las operaciones generalmente modifican las banderas de condición del ST en función del resultado ya sea éste 0, negativo, etc.. Este registro puede ser fácilmente almacenado y restaurado.
- **Registro de habilitación de interrupciones CPU/DMA (IE):** El IE es un registro de 32 bits. Los bits de habilitación de interrupciones del CPU están en las localidades 10-0, mientras que los de habilitación de interrupciones DMA están del 26-16. Un "1" habilita la interrupción correspondiente, mientras que un "0" la deshabilita.
- **Registro de bandera de interrupción del CPU (IF):** El IF es un registro de 32 bits, en el cual un "1" en el bit del registro IF indica que la interrupción correspondiente está habilitada y un "0" indica que determinada interrupción se encuentra deshabilitada.
- **Registro de banderas I/O (IOF):** Controla la función de las patillas externas, XF0 y XF1. Estos patillas pueden ser configurados para entradas o salidas, por lo que pueden ser leídas o sobre escritas.
- **Registro de dirección inicial de repetición (RS):** Cuando el procesador se encuentra operando en el modo de repetición, este registro contiene la dirección inicial del bloque de instrucciones a ser repetidas.
- **Registro de dirección final de repetición (RE):** Este registro contiene la última dirección del bloque del programa a ser repetido.
- **Contador de repetición (RC):** Este registro de 32 bits se utiliza para especificar el número de veces que un bloque de código será repetido cuando se realiza una repetición de bloque.
- **Contador de programa (PC):** El PC es un registro de 32 bits que contiene la dirección de la siguiente instrucción que será ejecutada por el CPU. Aunque el PC no forma parte del archivo de registro del CPU, es un registro que puede ser modificado por instrucciones que modifican el flujo del programa.

## 6.3.2 ORGANIZACIÓN DE LA MEMORIA

La capacidad total de memoria del TMS320C30 es de 16 Mpalabras de 32 bits. Dentro de éste espacio de direcciones se encuentran tanto programas, datos y espacios de entrada/salida, permitiendo que se almacenen tablas, coeficientes, códigos de programa o datos ya sea en memoria RAM ó en memoria ROM. De ésta forma, la memoria usada es maximizada y los espacios de memoria se pueden distribuir como se desee.

### 6.3.2.1 MEMORIA RAM, ROM Y CACHE

En la fig. 6.3 se muestra como se encuentra organizada la memoria en el TMS320C30. Los bloques de memoria RAM 0 y 1 son cada uno de 1k×32 bits. El bloque de memoria ROM es de 4k×32 bits. Cada bloque de RAM y ROM es capaz de soportar los dos accesos del CPU en un sólo ciclo. Debido a que los *buses* de programas, de datos y de DMA se encuentran separados, se pueden realizar en paralelo tanto accesos al código de programa, lecturas y escrituras de datos como operaciones DMA.

También se cuenta con un *cache* de instrucciones de 64×32 bits para almacenar secciones de código que se repiten constantemente, reduciendo el número de accesos externos que realiza el microprocesador. Los *buses* externos también son liberados para usarse por el DMA, accesos a memoria externa u otros dispositivos en el sistema.

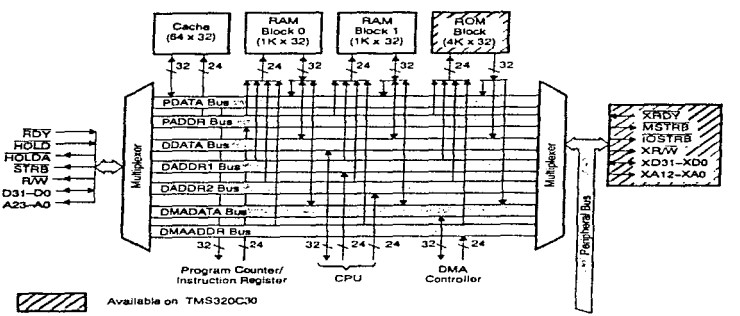



Fig. 6.3 Organización de la memoria del TMS320C30.

### 6.3.2.2 LOS MAPAS DE MEMORIA

El mapa de memoria del microprocesador se define a partir del modo en que este trabajando el TMS320C30, ya sea modo de microprocesador ( MC/$\overline{MP}$ = 0 ) o modo de microcomputadora ( MC/$\overline{MP}$ = 1 ). Los mapas de memoria para estos modos son similares y se muestran en la fig. 6.4. Las localidades 800000h – 801FFFh están mapeadas al *bus* de expansión, cuando se accesa a ésta región se activa el $\overline{MSTRB}$. Las localidades 8020000h – 803FFFh, son reservadas. Las localidades 804000h - 805FFFh están mapeadas al *bus* de expansión y cuando se accesa a esta región se activa el $\overline{IOSRTB}$. Las localidades 806000h - 807000h son reservadas. Todos los registros del *bus* de periféricos de memoria mapeada se localizan en 8080000h - 8097FFh. En ambos modos de trabajo, el bloque 0 de memoria RAM se localiza en la direcciones 809800h - 809BFFh, mientras que el bloque 1 de memoria se localiza en la 809C00h - 809FFFh. Las localidades 80A000h - 0FFFFFFh se accesan por el puerto de memoria externa ( $\overline{STRB}$ activo).

En modo microprocesador, la memoria ROM de 4 kbytes interna no está mapeada en el mapa de memoria. Las localidades 0h - 0BFh están destinadas a: vectores de interrupción, vectores de trampa y localidades reservadas, las cuales son accesadas por el puerto de memoria externo ( $\overline{STRB}$ activo). Las localidades 0C0h - 7FFFFFh también son accesadas por el puerto de memoria externo.

En modo microcomputadora, la memoria ROM de 4kpalabras interna se mapea en las localidades 0h - 0FFFh. Existen 192 localidades (0h-0BFh) dentro de este bloque para vectores de interrupciones, vectores de trampa y espacios reservados. Las localidades 1000h - 7FFFFFh se accesan por el puerto de memoria externa.

### 6.3.2.3 MODOS DE DIRECCIONAMIENTO DE LA MEMORIA

En el TMS320C30 se proporcionan cinco grupos de modos de direccionamiento y seis tipos de direccionamiento disponibles para los modos. Estos modos y tipos de direccionamiento son los siguientes:

- Modos de direccionamiento general:
  - ◊ Registro: el operando es un registro del CPU.
  - ◊ Corto inmediato: el operando es un valor inmediato de 16 bits.
  - ◊ Directo: el operando es el contenido de una dirección de 24 bits.
  - ◊ Indirecto: Un registro auxiliar indica la dirección del operando.
- Modos de direccionamiento de tres operandos:
  - ◊ Registro: de igual forma que el modo de direccionamiento general.
  - ◊ Indirecto: de igual forma que el modo de direccionamiento general.

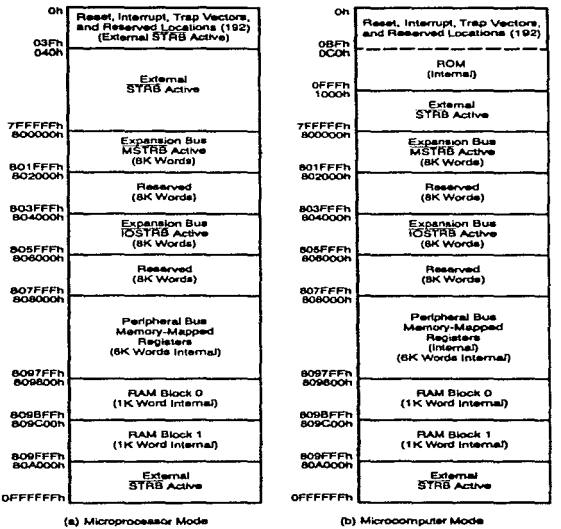



Fig. 6.4 Mapas de memoria para el TMS320C30.

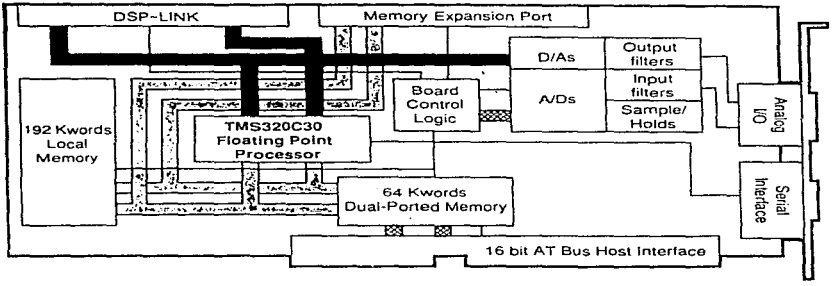
- Modos de direccionamiento paralelo:
  - ◊ Registro: el operando es un registro de precisión extendida.
  - ◊ Indirecto: de igual forma que el modo de direccionamiento general.
- Modo de direccionamiento largo inmediato:
  - ◊ El operando largo inmediato es un valor inmediato de 24 bits.
- Modos de direccionamiento de saltos condicionales:
  - ◊ Registro: de igual forma que el modo de direccionamiento general.
  - ◊ Relativo PC: Se suma un desplazamiento de 16 bits con signo al PC.

### 6.3.3 DESCRIPCIÓN GENERAL DEL P/N 600-00545

El desarrollo del sistema de cancelación de interferencia se llevo a cabo en una tarjeta de sistema para el TMS320C30 de SPECTRUM©, P/N 600-00545. El *software* utilizado para el desarrollo del programa para la tarjeta, fue el *TI C Compiler*. La tarjeta de sistema tiene un conector de entrada completo para PC-AT[10], que contiene dos conectores DB-15 para señales analógicas. En la fig. 6.5 se muestra la distribución de los componentes que integran esta tarjeta.

El P/N 600-00545 trabaja en modo microprocesador. El mapa de memoria par el P/N 600-00545 se presenta en la fig. 6.6. Este, contiene 4 bancos de memoria en la tarjeta, dos de los cuales (banco 0 y 3) se encuentran siempre ocupados. Las 64 kpalabras en el banco 0 es memoria de estado de espera cero que incluye las localidades del RESET y otras interrupciones. El banco 3 es un puerto *dual* entre la PC y el TMS320C30 y se compone con memoria de estado de espera "uno". Normalmente no se tendrá acceso a la memoria de los bancos 1 y 2, al menos que se tenga una opción de memoria extendida.

[10] PC-AT (Personal Computer Advanced Technology) : Computadora personal de tecnología avanzada.




- **Processor**
  TMS320C30 running at a 33.33 MHz clock rate (60ns instruction cycle).
  32-bit floating point arithmetic.
  Eight 40-bit accumulators.
  2 Kwords dual access RAM.
  4 Kwords dual access ROM.

- **Memory**
  Dual port: 64K X 32, 1 wait-state (35ns) RAM, dual ported between the C30 and the PC bus.
  C30 Local: 64K X 32, 0 wait-state (25ns) RAM, expandable to 192K X 32.
  Off Board: Memory expansion connector facilitates off-board expansion to full C30 capacity.

- **Analog Input**
  Two 16-bit ADCs with sample-and-hold, 150KHz throughput [Burr-Brown PCM78 (5us), Burr-Brown SHC5320 (1.5us)].
  Fourth order filters.
  Double-buffered data latching.
  Voltage range: ± 3V.
  Connections: DB15 style

- **Analog Output**
  Two 16-bit DACs [Burr-Brown PCM56 (1.5us)].
  Fourth-order filters.
  Double-buffered data latching.
  Voltage range: ± 3V.
  Connections: DB15 Style.

- **Analog I/O Clocking Options**
  Software initiated conversions.
  C30 on-chip timer initiated conversions
  External trigger pulses.

- **Serial I/O**
  Two synchronous 8.3 Mbps serial ports.
  All signals buffered.
  Fully configurable for global compatibility.

- **Interface to PC Host**
  16-bit PC/AT card format.
  Occupies 16 contiguous I/O ports (fully mappable).
  Choice of six PC interrupts.

- **DSP-LINK System Expansion Interface**
  16-bit parallel expansion bus.
  Up to 5 Mwords/sec transfer rate.
  8K memory-mapped DSP I/O ports available.
  Standard 50-pin male ribbon cable connector.
  DSP-LINK specifications are available for interfacing application-specific hardware.

- **Physical**
  Full length PC/AT plug-in board.
  Dimensions: 13 3/8"L x 4 5/8"H x 5/8"D.

- **Electrical**
  Power consumption: 5V @ 2.0A, 12V @ 150 mA, -12V @ 150mA

Fig. 6.6 Distribución de componentes que integran la tarjeta P/N 600-00545.

| Banco | Tamaño (palabras) | Wait State | Dirección(hex) |
|---|---|---|---|
| 0<br>Área A | 64 k | 0/1 | 000000<br>a 00FFFF |
| 1<br>Área A | 64 k | 0/1 | 010000<br>a 01FFFF |
| 2<br>Área A | 64 k | 0/1 | 020000<br>a 02FFFF |
| 3<br>Área B | 16 ó 64 k | 0/1 | 030000<br>a 03FFFF |
| Conector de<br>Expansión de<br>Memoria | 7936 k | Definida por el<br>Usuario | 0400000<br>a 7FFFFF |
| DSPLINK | 8 k | 2 | 800000<br>a 801FFF |
| Reservado | 8 k | | 802000<br>a 803FFF |
| Entrada/Salida<br>Analógica e<br>Interrupciones PC | 8 k | 2 | 804000<br>a 805FFF |
| Reservado | 8 k | | 806000<br>a 807FFF |
| Periféricos<br>dentro del Circuito<br>integrado | 6 k | Interno | 808000<br>a 8097FF |
| RAM 0 | 1 k | Interno | 809800<br>a 809BFF |
| RAM 1 | 1 k | Interno | 809C00<br>a 809FFF |
| Conector de<br>Expansión de<br>Memoria | 8152 k | Definida por el<br>Usuario | 80A000<br>a FFFFFF |

Fig. 6.6 Mapa de memoria de la tarjeta P/N 600-00545.

### 6.3.3.1 INTERFACES ANALÓGICAS DEL P/N 600-00545

La interfaz para señales analógicas del P/N 600-00545 contiene dos canales de entrada y dos de salida, estos canales se etiquetan con la letra A y B. Se tiene acceso a esta interfaz a través de tres registros de 16 bits que están conectados al *bus* de expansión del TMS320C30, sus direcciones se encuentran entre la dirección 804000h y 804008h. En la Fig. 6.7 se muestran las direcciones de estos registros.

| Función | Dirección |
|---|---|
| Canal A de lectura/escritura ADC y DAC | 804000 |
| Canal B de lectura/escritura ADC y DAC | 804001 |
| Generate Software Conversion Trigger | 804008 |

Fig. 6.7 Mapeo de la interfaz analógica.

Los primeros dos registros accesan a los convertidores ADC y DAC a los dos canales de entrada/salida analógicos. Los ADC y DAC se comunican con los datos en un formato de complemento a 2 de 16 bits. La dirección 804008h accesa a un registro que puede usarse para comenzar una conversión del ADC y DAC. Todos los registro se accesan con dos estados de espera de memoria, resultando un tiempo total de 180 ns.

Los convertidores ADC y DAC usan el mismo registro para entrada y salida. El valor del ADC se transfiere al registro con el contenido del registro previo y es simultáneamente transferido al DAC sin ninguna intervención del procesador. Por lo tanto, la señal analógica por *default*, en cada canal, será repetida directamente al canal de salida analógica correspondiente con retardo de una muestra.

En la fig. 6.8 se muestra el conector y las funciones correspondientes a cada patilla se dan a continuación:

1. Salida del *buffer* canal A
2. Salida DAC canal A
3. Salida del *buffer* canal B
4. Salida del DAC canal B
5. Entrada al ADC canal A
6. Monitor S/H canal A
7. Entrada al ADC canal B
8. Monitor S/H canal B
9. Tierra del canal A
10. Tierra común
11. Tierra del canal B
12. Reloj del *trigger* externo
13. Tierra canal A
14. Tierra común
15. Tierra canal B

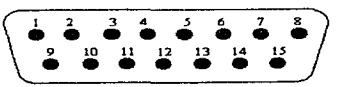



Fig. 6.8 Conector DB-15 para la interfaz analógica.

Si se desea leer un valor del DAC y escribir un valor diferente al DAC sobre el mismo canal durante el mismo intervalo de muestra, se debe leer primero el ADC antes de escribir en el DAC.

### 6.3.4 SOFTWARE PARA EL CONTROL DEL TMS320C30

En la fig. 6.9 se muestra el flujo para el desarrollo de software del TMS320C30, siendo más común el camino que se muestra en líneas discontinuas. Siguiendo esta trayectoria, los pasos a seguir son los siguientes:

- El **compilador C** acepta un código fuente escrito en lenguaje C y produce un código fuente en lenguaje ensamblador para el TMS320C3x/C4x. Se incluye también en el compilador un programa *shell* y la utilidad de interlistado. El programa *shell* permite automáticamente compilar, ensamblar y ligar módulos de fuentes, mientras que la utilidad de interlistado, interlista el código fuente en C con la salida en lenguaje ensamblador.
- El **ensamblador** traduce los archivos fuente de lenguaje ensamblador en archivos objeto COFF[11] de lenguaje máquina. El formato COFF hace que la programación modular sea más fácil porque estimula a pensar en términos de bloques de código y datos cuando se escriben programas en ensamblador. A estos bloques se les conoce como secciones y tanto el ensamblador como el *linker* proporcionan directivas que permiten crear y manipular secciones. Si el lector desea conocer más detalles de este formato, puede consultar la guía de usuario del ensamblador en [Ref. 7].
- El **archivador** permite coleccionar un grupo de archivos en uno solo conocido como librería.
- Las dos librerías objeto **rts30.lib** y **rts40.lib** se unen con el compilador C. Estas librerías contienen funciones estándar de ejecución, funciones de utilidad para el compilador y funciones matemáticas que pueden ser llamadas por los programas en C que han sido compiladas para el TMS320C3x o el TMS320C4x.
- El **linker** fusiona los archivos objeto en un solo módulo objeto ejecutable. Al crearse el módulo objeto, el *linker* lleva a cabo la relocalización de variables y resuelve las referencias externas. El *linker* acepta como entrada archivos objetos COFF relocalizables y librerías objeto.

#### 6.3.4.1 CARACTERÍSTICAS DEL COMPILADOR C TMS320 DE PUNTO FLOTANTE

Las características mas sobresalientes de este compilador se enuncian a continuación:

- **Estándar ANSI C:** El compilador C TMS320 de punto flotante obedece al estándar ANSI C definido por la especificación ANSI y descrito en [Ref. 1].

[11] COFF (Common Object File Format) : Formato común de archivo objeto.

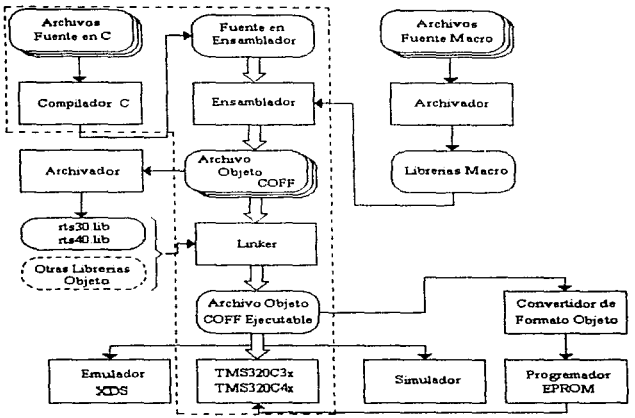



Fig. 6.9 Flujo para el desarrollo de software para el TMS320C3x/C4x.

- **Optimización:** El compilador permite realizar una optimización sofisticada que emplea diversas técnicas avanzadas para generar un código eficiente y compacto de la fuente en C. La optimización general puede emplearse a cualquier código en C, mientras que la optimización específica usa los beneficios de la arquitectura del TMS320C3x/C4x.
- **Salida de fuente de lenguaje ensamblador:** El compilador genera una fuente de salida en lenguaje ensamblador que puede inspeccionarse fácilmente, permitiendo ver el código en ensamblador de los archivos fuente en C.
- **Modelos de memoria grande y pequeña:** El compilador soporta dos modelos de memoria. El modelo de memoria pequeña permite al compilador accesar a la memoria restringiendo los espacios de datos globales a una sola página de datos con 64 kpalabras. El modelo de memoria grande permite un espacio ilimitado.
- **Programa shell del compilador:** El paquete del compilador incluye un programa *shell* que permite compilar, ensamblar y ligar programas en un solo paso.
- **Preprocesador integrado:** El preprocesador C se encuentra integrado con el *parser*, para una compilación más rápida.

- **Archivos objeto COFF:** El formato de archivo objeto común permite definir el mapa de memoria del sistema al momento de ligar. Esto maximiza el desempeño al permitir ligar código en C y objetos de datos en áreas de memoria específica.
- **Utilidad de interlistado:** El paquete del compilador contiene una utilidad que permite interlistar las sentencias de la fuente en C original en la salida de lenguaje ensamblador del compilador. Esta utilidad proporciona un método para examinar el código ensamblador generado para cada sentencia del programa en C.
- **Tamaño de datos de 32 bits:** Todos los tamaños de los datos son de 32 bits. Esto hace que todos los tipos de datos aprovechen las capacidades aritméticas de tipo entero y punto flotante del TMS320C3x/C4x.

El programa *shell* del compilador realiza la tarea de compilar, ensamblar y ligar los archivos fuente en C para crear un archivo ejecutable COFF con un solo comando llamado "cl30". El *shell* "cl30" corre uno o más módulos fuente de la siguiente manera:

- El **compilador** incluye el *parser*, optimizador (opcional) y el generador de código.
- El **ensamblador** genera un archivo objeto COFF.
- El **linker** (opcional) liga los archivos para crear un archivo objeto ejecutable. Hasta este momento el proceso de ligado es opcional. Se puede compilar y ensamblar varios archivos con el cl30 y ligarlos posteriormente.

Por *default*, el cl30 compila y ensambla archivos, sin embargo, si se usa la opción -z, cl30 también realizará el proceso de ligado. En la fig. 6.10 se muestra el proceso que sigue el comando cl30.

La manera de invocar el programa *shell* es:

```
cl30 [-opciones] [nombre_archivos] [-z [opciones_ligado]]
```

donde:

**cl30** : es el comando que invoca al compilador y al ensamblador.

opciones: influyen sobre la manera en que el compilador procesa los archivos de entrada.

nombre_archivos : son uno o más archivos fuente en C, archivos fuente en ensamblador o archivos objeto.

-z : es la opción que ejecuta el *linker*.

opciones_ligado : controla los procesos de ligado.

Las opciones controlan la forma en que el compilador procesa los archivos. Los nombres de los archivos proporcionan un método para identificar los archivos fuente, archivos inmediatos y archivos de salida. Las opciones y los nombres de los archivos pueden escribirse en cualquier orden en la línea de comando, sin embargo, si se usa la opción -z, ésta y la información asociada debe seguir a todos los nombres de archivo y las opciones de compilación.

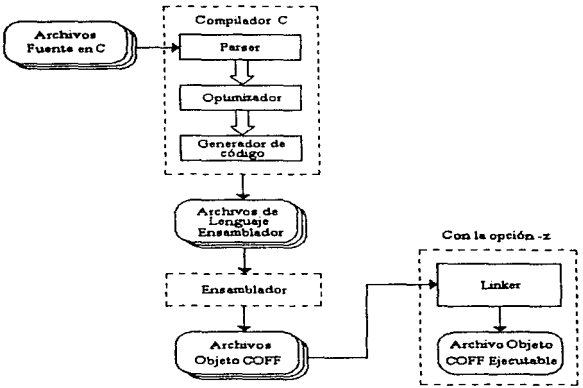



Fig. 6.10 Proceso que sigue el programa *shell* cl30.

Los archivos de entrada descritos en la línea de comando deben ser archivos fuente en C, archivos fuente en ensamblador o archivos objeto. El comando "cl30" identifica el tipo de archivo por su extensión, de la siguiente manera:

| **Extensión** | **Tipo de archivo** |
|---|---|
| .asm ó .s* (la extensión comience con s) | Archivo fuente en ensamblador |
| .c ó sin extensión | Archivo fuente en C |
| .o* (la extensión comience con o) | Archivo objeto |

Se puede consultar la guía del usuario del compilador C de la [Ref. 8] en la que se describen las opciones disponibles del "cl30" y del *linker*.

### 6.3.4.2 CONTROL DE LOS PROGRAMAS DEL TMS320C30 MEDIANTE SOFTWARE PARA PC

El P/N 600-00545 incluye una librería que permite escribir programas para IBM PC en un lenguaje de alto nivel. Las rutinas de esta librería se usan para cargar código objeto del TMS320C30 a una o más tarjetas, empezar la ejecución del código en la tarjeta, detener la ejecución y transmitir datos de la PC a la tarjeta y viceversa.

Las rutinas de esta librería son llamadas más fácilmente por programas escritos en Microsoft C, pero se pueden usar en cualquier lenguaje de programación de Microsoft. El archivo fuente que contiene la librería C es el "30LIBRAR.C", sin embargo, al compilar se requiere diferenciar entre un modelo de memoria pequeño o grande. La librería fuente C está precompilada en los archivos "SM30DEV.LIB" y "LM30DEV.LIB" para modelos de memoria pequeña y grande respectivamente.

El proceso que se debe seguir para crear un programa usando las funciones de la librería es el siguiente:

1) Incluir el encabezado "TMS30.H" en el archivo fuente en C. Esto hace que se declaren todas las funciones de la librería usando los prototipos de las funciones. El lector puede consultar el manual de usuario de la tarjeta de sistema TMS320C30 de SPECTRUM© como referencia para las funciones disponibles en la librería.

2) Usar la función **SelectBoard()** para iniciar la interfaz de la PC a la tarjeta. En particular la dirección de la interfaz de esta tarjeta es 390h, por lo que la función debe escribirse **SelectBoard(0x390)**.

3) Cargar a al tarjeta el programa de código objeto del TMS320C30, usando la función **LoadObjectFile()**. Si solo se desea escribir y leer datos en la memoria de la tarjeta, entonces, este paso es opcional.

4) Usar las funciones de transferencia de datos entre la PC y la tarjeta.

5) Compilar el programa, ligarlo con el archivo objeto de la librería "*.LIB". Otra forma de hacerlo es compilar el archivo "30LIBRAR.C" y ligar el archivo objeto resultante con el archivo objeto del programa de control.

Solo se puede tener acceso al banco 3 de memoria sin detener la ejecución del procesador. Debido a que algunas funciones de la librería requieren un parámetro que especifica el tipo de acceso a la memoria, se debe declarar "DUAL", de lo contrario, el procesador se detendrá para accesar a la memoria.

# Capítulo 7

## SISTEMA DE CANCELACIÓN ADAPTIVA DE INTERFERENCIA

El ruido es una palabra que tiene una connotación negativa e implica algo no deseado, éste puede presentarse en algunos fenómenos de forma acústica, eléctrico o incluso óptico. A su vez se puede clasificar de diversas maneras, puede ser continuo como un ronroneo; aleatorio, como el revoloteo de las hojas de un árbol o transitorio como el estruendo de un rayo. El interés de conocer los efectos que éste puede provocar en el ser humano, son considerados en muchas áreas en donde el medio ambiente del hombre es importante para su desenvolvimiento. Se han descubierto diversos efectos sobre éste que pueden ser auditivos, fisiológicos o psicológicos. La inteligibilidad de señales audibles suele ser de bastante importancia en algunos medios comunicación y si ésta se ve afectada por una señal de ruido se considera un efecto negativo, por ejemplo, si no se tiene una buena inteligibilidad de una conversación telefónica no se puede establecer una buena comunicación. Por otro lado, existen señales de ruido que pueden producir efectos negativos sobre los sistemas que ya he venido comentando, por ejemplo, los efectos que puede causar la señal de 60 Hz de la línea eléctrica que interfiere en una sala quirúrgica pueden ser de mucha importancia. El tratamiento para una posible solución en la eliminación de una señal de interferencia sobre cualquier señal debe ser el mismo. Un medio eficaz para la solución de estos problemas es el filtrado. Los filtros pueden ser fijos o adaptivos. Si se trata de filtros fijos, el diseño de estos se basa en el conocimiento previo de la señal de ruido, mientras que los filtro adaptivos poseen la habilidad de ajustar sus propios parámetros automáticamente, por lo que su diseño requiere de poco o nada del conocimiento de las características del ruido. En este capítulo se describe el desarrollo del sistema de cancelación de interferencia basado en un filtro adaptivo en la tarjeta P/N 600-00545.

### 7.1. DESCRIPCIÓN DEL SISTEMA

El interés de esta tesis es desarrollar un sistema adaptivo para la cancelación de interferencia sólo para lograr una buena inteligibilidad de las señales que se ven afectadas por el ruido. El cancelador de interferencia es una variación de un filtro óptimo. El sistema requiere una

señal de referencia derivada de alguna forma del ruido, que será aplicada al filtro adaptivo para que la salida pueda sustraerse de la señal primaria que contiene tanto el ruido como la señal de interés, de esta forma la señal de ruido será atenuada o eliminada por el cancelador. Si la sustracción del ruido de la señal primaria se realiza inapropiadamente, como resultado se puede tener un incremento en la potencia del ruido. En la fig. 7.1 se observa el diagrama de bloques del sistema de cancelación adaptivo.

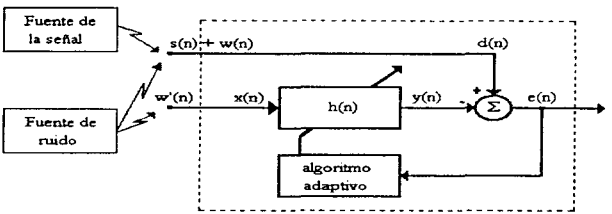



Fig. 7.1. Diagrama de bloques del sistema de cancelación adaptivo de interferencia.

Un sensor recibe la señal s(n) y w(n), ésta última es la señal de ruido que se desea eliminar. Las señales combinadas s(n) + w(n), forman la entrada primaria al sistema. Un sensor más recibe una versión del ruido w'(n), esta señal no está correlacionado con la señal s(n), pero de algún modo si lo está con el ruido w(n). La señal w'(n) es filtrada para producir y(n) que intenta ser una réplica de w(n). La salida del filtro se sustrae de la entrada primaria, es decir, $e(n) = s(n) + w(n) - y(n)$ para que e(n) resulte solo s(n). La señal e(n) se usa como señal de retroalimentación y se aplica al algoritmo LMS, *vid. supra* p. 45, este algoritmo ajusta los valores de los coeficientes del filtro para minimizar el error.

En muchos casos, se tiene una señal de banda ancha s(n) y la señal de ruido w(n) suele ser una señal periódica y no se dispone de una señal de referencia. Pues bien, como se muestra en la fig. 7.2, se puede aplicar un retraso a la señal de entrada primaria para usarse como señal de referencia, de tal manera que el retardo decorrelacione la señal de banda ancha; así, la señal periódica en muchos casos puede ser fácilmente cancelada. Es importante considerar que el retraso sea adecuado para lograr la decorrelación de los componentes de la señal de banda ancha, mientras que los componentes de la señal de interferencia, por su naturaleza periódica, permanecerán correlacionados.

El sistema de cancelación de interferencia diseñado para esta tesis funciona bajo las dos situaciones descritas anteriormente: (1) el sistema recibe una señal de referencia y (2) el sistema no recibe la señal de referencia, pero crea mediante un retraso esta señal.

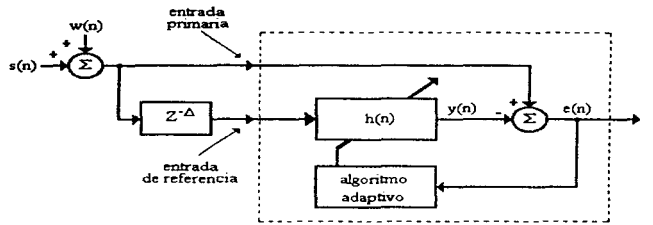



Fig. 7.2 Cancelador adaptivo de interferencia periódica.

## 7.2. SIMULACIÓN DEL SISTEMA EN MATLAB©

Para lograr un mejor diseño del sistema en el DSP, desarrollé un programa en Matlab© (que no presento aquí) para simular su funcionamiento. En la fig. 7.3 se muestra el diagrama de un cancelador de interferencia. Las señales que se usaron para la simulación son: una secuencia de números aleatoria distribuida como señal de interés y una señal cuadrada de 60 Hz como señal de interferencia. El usó de la señal cuadrada como interferencia se debe a que contiene una gran cantidad de armónicos múltiplos de la frecuencia fundamental. Así pues, las señales son:

s(n) = una secuencia de números aleatorios distribuida que representa una señal de banda ancha.

w(n) = una señal cuadrada de 60 Hz.

Estas señales se filtraron por un filtro pasa-bajas a nivel *software* en el programa de Matlab© con frecuencia de corte de 750 Hz[1]. Los datos adicionales para el filtro son:

Frecuencia de muestreo = 6000 Hz.

Número de muestras = 600

Número de coeficientes del filtro = 100

Constante de adaptación del algoritmo LMS = 0.0008

[1] El proceso de filtrado que realiza Matlab se basa en un algoritmo de programas para procesamiento digital de señales de IEEE. Este algoritmo usa la forma directa II transpuesta de un filtro IIR, por lo que se espera un defase y alteración de la señal.

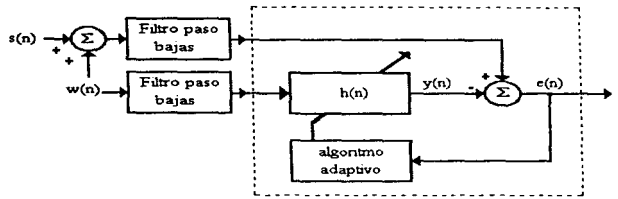



Fig. 7.3 Diagrama del cancelador de interferencia.

En la fig. 7.4a se muestran las señales de entrada al sistema en el dominio del tiempo, la línea continua representa la señal s(n), mientras que la línea discontinua representa la señal de interferencia w(n). Se suman éstas señales para obtener la entrada primaria al filtro adaptivo, d(n). La señal w(n) será la señal de entrada de referencia al filtro.

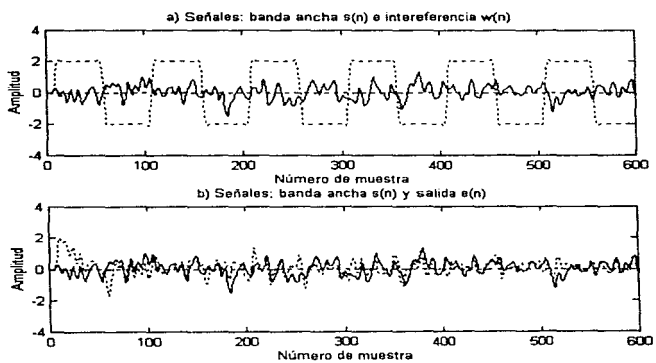



Fig. 7.4 a) Señales originales; b) Comparación de la señal de banda ancha con la salida del sistema.

En la fig. 7.4b se comparan las señales: de banda ancha s(n) (línea continua) y salida del sistema e(n) (línea discontinua) en el dominio del tiempo. Es notable como la señal de salida del sistema e(n), logra parecerse mas a la señal s(n) a medida que transcurre el tiempo (número de muestras). Se puede estimar que si se procesan más muestras, la señal e(n) logre parecerse más a s(n). En la fig. 7.5a se tienen la señal de interferencia w(n) (línea continua) y la señal y(n) (línea discontinua) esta última es la salida del filtro adaptivo que intenta ser la réplica de la interferencia. Finalmente, en la fig. 7.5b, se muestra el valor de los coeficientes del filtro en su etapa final, es decir, después terminar el proceso.

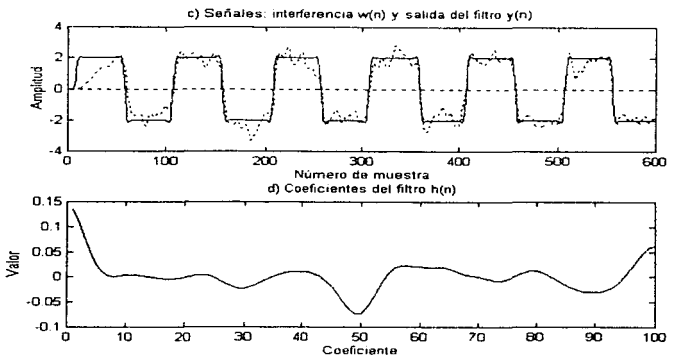



Fig. 7.5 a) Comparación de la señal de interferencia w(n), línea continua con la obtenida por el filtro adaptivo y(n), línea discontinua y b) Coeficientes del filtro en su etapa de adaptación.

En la fig. 7.6 se presentan los resultados del proceso en el dominio de la frecuencia. En 7.6a se presentan los espectros de la señal de entrada primaria d(n) (línea continua) y la señal resultante después del filtrado (línea discontinua). Como se puede observa, el espectro d(n) presenta algunos "picos" que corresponden a los componentes de la señal de interferencia, el "pico" más alto se presenta en la frecuencia de 60 Hz, que es la frecuencia fundamental de las señal cuadrada. Los "picos" más pequeños representan los armónicos de esta señal y coinciden en frecuencias múltiplos a 60 Hz. Después del filtrado es posible atenuar en gran manera la magnitud de estos "picos", pero también se llega a perder un poco la señal de interés. En 7.6b se comparan las señales s(n) (línea continua) con la señal de salida del sistema e(n) (línea discontinua) y se logra ver como estas señales no difieren demasiado entre sí.

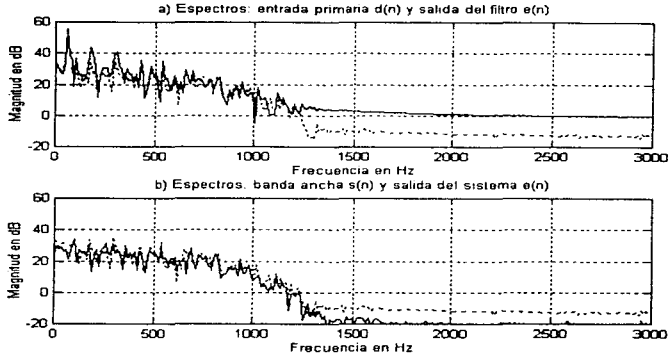



Fig. 7.6 a) Espectro de la señal primaria d(n) (línea continua) vs. señal recuperada del proceso de cancelación e(n) (línea discontinua). b) Espectro de la señal s(n) (línea continua) vs. espectro de la señal e(n) (línea discontinua).

## 7.3. DESARROLLO DEL SISTEMA EN TIEMPO REAL

El sistema de cancelación de interferencia adaptivo está compuesto por tres programas: un programa diseñado para trabajar en el DSP, un programa en ambiente DOS para controlar el programa del DSP y finalmente un programa gráfico para Windows© que maneja al programa para ambiente en DOS. En la fig. 7.7 se tiene un diagrama de bloques en el cual se muestra la relación existente entre estos programas.


Fig. 7.7 Relación entre los programas para el cancelador de interferencia.

Es importante hacer notar que los programas para el DSP ("anc.out") y el de ambiente DOS ("ancpc.exe") son independientes del programa para Windows©, es decir, se puede ejecutar "anc.out" mediante "ancpc.exe" sin necesidad de ejecutar el de Windows©. Los listados del

programa para el DSP y para ambiente DOS se presentan en el apéndice III y IV respectivamente. Aquí solo explicaré brevemente el funcionamiento de estos.

### 7.3.1 PROGRAMA "ANC.OUT" PARA FUNCIONAMIENTO EN LA TARJETA

El programa para el DSP está escrito en lenguaje C y compilado por el *TMS320 floating-point C compiler*. A partir de un archivo fuente "anc.c" que está escrito en lenguaje ANSI C se crea un archivo objeto COFF que posteriormente es ligado por *software* para obtener un archivo "anc.out" COFF ejecutable.

Para comprender el programa es necesario explicar los encabezados (*headers*) que se requieren.

#### C30LSI.H

El encabezado "c30lsi.h" se muestra en el apéndice V. En este encabezado se encuentran declarados los macros que permitirán leer el código referente a direcciones y constantes de la tarjeta. Este encabezado así como otros usados en los programas para la tarjeta fueron creados por el Dr. Felipe Orduña.

Las siguientes macros son direcciones de algunos registros del DSP:

```
#define C30_PBUS_CTRL          (*(long *)0x808064)
#define C30_XBUS_CTRL          (*(long *)0x808060)
#define C30_TIMER1_CTRL        (*(long *)0x808030)
#define C30_TIMER1_COUNT       (*(long *)0x808038)
```

Macros para definir constantes para los convertidores de la tarjeta:

```
#define C30_ADC_UNITS_PER_VOLT   7.158278826e+8
#define C30_ADC_BASE             0x804000
#define ADC(n)                   (long)(*(long *)(C30_ADC_BASE + (n)))
#define DAC(n)                   (*(long *)(C30_ADC_BASE + (n)))
```

Unidades de conversión del ADC y DAC:

```
#define C30_ADC_UNITS_PER_VOLT  7.158278826e+8
#define C30_DAC_UNITS_PER_VOLT  7.158278826e+8
#define C30_ADC_VOLTS_PER_UNIT  1.396983862e-9
#define C30_DAC_VOLTS_PER_UNIT  1.396983862e-9
```

Conversión para la frecuencia de muestreo a valores de conteo del TIMER:

```
#define C30_GET_COUNT(fs)     (unsigned long)(8.3333333e+6/(float)(fs))
```

Macro para el ajuste de la frecuencia de muestro al TIMER1:

```
#define C30_SET_TIMER1(fs)
{
  C30_PBUS_CTRL = 0x0800;
  C30_XBUS_CTRL = 0x0000;
  C30_TIMER1_CTRL = 0x0601;
  C30_TIMER1_COUNT = C30_GET_COUNT(fs);
  C30_TIMER1_CTRL = 0x06C1;
}
```

Prepara las llamadas a la interrupción 'n' a la dirección específica:

```
#define C30_SET_INTERRUPT(n,adrs)
{
  asm("  AND    00001FFFh,ST");
  (*(unsigned long *)(n + 1)) = (unsigned long)adrs;
}
```

Habilita la interrupción:

```
#define C30_ADC_START()
{
  asm("  OR   00000002h,IE");
  asm("  OR   00002000h,ST");
}
```

Deshabilita la interrupción:

```
#define C30_ADC_STOP()
{
  asm("  AND    0FFFFFFFDh,IE");
  asm("  AND    0FFFFDFFFh,ST");
}
```

El siguiente código de bit declara una etiqueta global al punto de entrada C "_c_int00" (definido en RTS30LIB) donde se inicializa el procesador y la función principal main() del usuario es llamada:

```
asm("  .sect  \".init\"");
asm("  .word  _c_int00");
asm("  .globl  _c_int00");
```

## ANC.H

"anc.h" es un encabezado que contiene macros para ser usados tanto por el programa de la tarjeta como por el programa de ambiente DOS. El encabezado completo se muestra en el apéndice VI.

Las siguientes macros definen los valores de inicio de las variables en la tarjeta:

| | | | |
|---|---|---|---|
| #define | SAMPLE_RATE_HZ | 10000.0 | Frecuencia de muestreo en Hz |
| #define | Filter_length | 32 | Tamaño del filtro adaptivo |
| #define | Shifter | 256 | Tamaño del vector de entradas |
| #define | MU | 1e-20 | Constante de adaptación |
| #define | DELAY | 0 | Muestras de retardo de la señal de entrada primaria |

Definición de las constantes para cada función del programa de la tarjeta:

| | | |
|---|---|---|
| #define | DSP_READY | 0 |
| #define | DSP_RESET | 1 |
| #define | DSP_INIT | 2 |
| #define | DSP_START | 3 |
| #define | DSP_STOP | 4 |
| #define | DSP_SET_ID | 5 |
| #define | DSP_UNSET_ID | 6 |
| #define | DSP_LAST | 7 |

Definición de estados de la tarjeta:

| | | | |
|---|---|---|---|
| #define | DSP_OFF | 0 | Tarjeta DSP inactiva |
| #define | DSP_ON | 1 | Tarjeta DSP activa |

Definición de presencia o ausencia de una señal de referencia:

| | | | |
|---|---|---|---|
| #define | REF_OFF | 0 | No se dispone de señal de referencia |
| #define | REF_ON | 1 | Se dispone alguna señal de referencia |

## ANC.C → ANC.OUT

El programa inicia con la declaración de las siguientes funciones disponibles en la tarjeta:

```
void dsp_reset (void);
void dsp_init (void);
void dsp_start (void);
void dsp_stop (void);
```

```
void dsp_set_id(void);
void dsp_unset_id(void);
```

Se declaran las variables usadas en el programa, algunas de estas toman los valores declarados en "anc.h":

```
int n = 0;                          Contador auxiliar
int id = 0;                         Identificador del programa
int Sh = Shifter;                   Tamaño de registro para almacenamiento de valores de
                                    entrada
int Fl = Filter_length;             Tamaño del filtro adaptivo
int ref = REF_OFF;                  Asignación de referencia disponible
int state = DSP_OFF;                Representa el estado del filtro (activo/inactivo)
int delay = DELAY;                  Indica el número de retardos de la señal de entrada
float (*H) = NULL;                  Apuntador hacia el vector de coeficientes del filtro
float (*X) = NULL;                  Apuntador hacia el vector de entradas
float mu = MU;                      Constante de adaptación del filtro adaptivo
float rate = SAMPLE_RATE_HZ;        Frecuencia de muestreo de los convertidores
volatile int dspMessage;            Recibe orden del programa desde la PC.
```

La siguiente función reinicia a la tarjeta con los valores declarados en "anc.h":

```
void dsp_reset ()
{

 mu = MU;
 delay = DELAY;
 rate = SAMPLE_RATE_HZ;

 C30_SET_INTERRUPT(1,c_int01);      Prepara una rutina de interrupción en la tarjeta
 C30_SET_TIMER1(rate);              Fija al TIMER1 de la tarjeta con la frecuencia de
                                    muestreo asignada en la variable rate
 C30_ADC_START();                   Inicia la rutina de interrupción
 state = DSP_ON;                    Cambia la variable de estado del DSP
}
```

La siguiente función reserva un bloque de memoria para los vectores de coeficientes del filtro y de entrada al filtro:

```
void dsp_init ()
{
 n = 0;
 if (H != NULL) free (H);
 H = calloc (Fl, sizeof (float));
 if (X != NULL) free (X);
 X = calloc (Sh, sizeof (float));
```

```
}
```

Esta función activa la rutina de interrupción en la tarjeta:

```
void dsp_start ()
{
 C30_SET_INTERRUPT(1,c_int01);
 C30_SET_TIMER1(rate);
 C30_ADC_START();
 state = DSP_ON;
}
```

La siguiente rutina detiene la ejecución de la tarjeta y libera el bloque de memoria asignado para los vectores de coeficientes del filtro y de entrada:

```
void dsp_stop ()
{
 C30_ADC_STOP ();                Detiene la rutina de interrupción
 C30_SET_TIMER1 (0);             Asigna el valor de cero al TIMER1
 free (H);                       Libera la memoria asignada para H
 free (X);                       Libera la memoria asignada en DelayX
 state = DSP_OFF;                Asigna el valor de cero a la variable de estado
}
```

Las dos rutinas siguientes trabajan conjuntamente para identificar desde el *prompt* de la PC si el programa que se encuentra actualmente cargado en la tarjeta es "anc.out":

```
void dsp_set_id ()              Fija a uno a la variable id
{
 id = 1;
}

void dsp_unset_id ()            Fija a cero la variable id
{
 id = 0;
}
```

Este es el programa principal:

```
void main (void)
{
 for (;;)                       Ciclo infinito
 {
  dspMessage = DSP_READY;       Asigna a la variable que recibe los mensajes de la PC
                                en un estado de espera
  while (dspMessage == DSP_READY);
```

```
handle [dspMessage] ();                    Recibe el nuevo mensaje de la PC y salta a la rutina
                                           indicada.
}
}
```

Esta es la rutina de interrupción que realiza el proceso de filtrado:

```
void c_int01()
{
float Primary_input;                       Almacena la señal de entrada del filtro en un instante
float y;                                   Almacena la señal de salida del filtro en un instante
float e, e_mu;                             Variables que almacenan el error y el error por la cte. mu
float *px;                                 Apuntador hacia el vector de entrada
float *ph;                                 Apuntador hacia el vector de coeficientes del filtro
int A1, A2;                                Contadores auxiliares
int k, l;                                  Contadores auxiliares

Primary_input = ADC(0);                    Almacena en la variable Primary_input el valor actual del
                                           ADC canal A de la tarjeta
if (ref == REF_OFF)                        Si no se dispone de señal de referencia o al menos el usuario
{                                          no la declaró, entonces retarda la señal recibida en el ADC
A1 = Sh;                                   canal A el número de muestras indicadas en "delay" y la
A2 = A1 - Fl;                              almacena en el vector X
n = (n - 1) & (A1 - 1);
X[n] = Primary_input;
l = (n - delay) & (A1 - 1);
}
else                                       Se dispone la señal de referencia, por lo que esta señal
{                                          es recibida a través del ADC canal B de la tarjeta y
A1 = Fl;                                   almacenada en el vector X
A2 = 0;
n = (n - 1) & (A1 - 1);
X[n] = ADC(1);
l = n;
}
```

El siguiente código del programa se encarga de realizar la suma de convolución entre el vector de entradas y los coeficientes del filtro, éste se ha implementado de tal forma que el apuntador que recorre al vector de entradas lo accesa como si éste fuera circular, con el fin de evitar el corrimiento de los valores del vector cuando se recibe un nuevo dato, reduciendo así el tiempo de trabajo del procesador.

```
y = 0.0;
 if (l >= A2)
 {
 k = A1 - l;
 px = X + l;
 ph = H;
 while (k--) y += *px++ * *ph++;
 k = l - A2;
 px = X;
 while (k--) y += *px++ * *ph++;
 }
 else
 {
 px = X + l;
 ph = H;
 k = Fl;
 while (k--) Filter_output += *px++ * *ph++;
 }
```

La siguiente rutina actualiza los valores de los coeficientes del filtro. El criterio usado para la actualización de los vectores es el algoritmo LMS, *vid. infra* p. 45. Una vez obtenida la respuesta del filtro del diagrama de bloques para un sistema de cancelación de interferencia, *vid. infra* p. 77 que se almacena en "y", se calcula el error producido por el valor deseado "Primary_input" y el valor de "y". Este error para un cancelador de interferencia no es otra cosa que la señal de interés s(n). Para actualizar los coeficientes del filtro se requiere el error multiplicado por la constante de adaptación:

```
e = Primary_input - y;          Diferencia producida entre d(n) y y(n)
e_mu = e * mu;                  Error multiplicado por la cte. de adaptación
if (l >= A2)
{
k = A1 - l;
px = X + l;
ph = H;
while (k--) *(ph++) += e_mu * *px++;
k = l - A2;
px = X;
while (k--) *(ph++) += e_mu * *px++;
}
else
{
px = X + l;
ph = H;
k = Fl;
while (k--) *(ph++) += e_mu * *px++;
}
```

Finalmente, los valores resultantes: error (señal recuperada) y salida del filtro (interferencia) durante el proceso son enviados a los DAC canal A y canal B respectivamente.

```
DAC(0) = e;
DAC(1) = y;
}
```

### 7.3.2 PROGRAMA "ANCPC.C" PARA EL CONTROL DEL PROGRAMA EN LA TARJETA

"ancpc.c" es un programa para el control de "anc.out" mediante la PC, codificado lenguaje C y compilado por Microsoft Visual C++©. Este programa comienza su ejecución escribiendo en el *prompt* de DOS su propio nombre y una línea de caracteres. El programa ya compilado y ligado se llama "ancpc.exe", este programa contiene una pequeña ayuda que aparecerá en pantalla si se teclea la siguiente línea de comando desde el *prompt* de DOS.

Esta pantalla de ayuda proporciona la información requerida acerca de las opciones disponibles de "anc.out". Para hacer funcionar a "ancpc" se tienen las siguientes opciones: "ancpc [ ... ]" pone en funcionamiento al programa "anc.out" con los parámetros indicados dentro de los corchetes. "ancpc -l" carga a la memoria de la tarjeta a "anc.out". "ancpc -s" detiene la ejecución del programa "anc.out" en caso de que este se encuentre en funcionamiento. "ancpc -e" produce un *reset* a la tarjeta. "ancpc -t" crea un archivo llamado "outdata" que contiene los valores de los parámetros del sistema. "ancpc -h" despliega en pantalla la ayuda.

```
This is ANCPC program Version 1.0 PC Handler
Usage:
        ancpc [-f f_length -r rate -m mu -d delay -n -v]
        ancpc -l
        ancpc -s
        ancpc -e
        ancpc -t
        ancpc -h

-f f_length : filter length, default 32
-r rate     : sampling rate, default 10kHz
-m mu       : adaptive constant, default 1e-20
-d delay    : delay samples, default 0
-n : reference option, receive reference input
-l : load DSP program to the board
-s : stop execution
-e : reset to the board
-t : dsp program test
-v : verbose option, print details during execution
-h : help and exit
```

Cuando el programa se ejecuta por primera vez, se debe usar la opción "ancpc -l"

```
DAC(0) = e;
DAC(1) = y;
}
```

### 7.3.2 PROGRAMA "ANCPC.C" PARA EL CONTROL DEL PROGRAMA EN LA TARJETA

"ancpc.c" es un programa para el control de "anc.out" mediante la PC, codificado lenguaje C y compilado por Microsoft Visual C++©. Este programa comienza su ejecución escribiendo en el *prompt* de DOS su propio nombre y una línea de caracteres. El programa ya compilado y ligado se llama "ancpc.exe", este programa contiene una pequeña ayuda que aparecerá en pantalla si se teclea la siguiente línea de comando desde el *prompt* de DOS.

Esta pantalla de ayuda proporciona la información requerida acerca de las opciones disponibles de "anc.out". Para hacer funcionar a "ancpc" se tienen las siguientes opciones: "ancpc [ ... ]" pone en funcionamiento al programa "anc.out" con los parámetros indicados dentro de los corchetes. "ancpc -l" carga a la memoria de la tarjeta a "anc.out". "ancpc -s" detiene la ejecución del programa "anc.out" en caso de que este se encuentre en funcionamiento. "ancpc -e" produce un *reset* a la tarjeta. "ancpc -t" crea un archivo llamado "outdata" que contiene los valores de los parámetros del sistema. "ancpc -h" despliega en pantalla la ayuda.

```
This is ANCPC program Version 1.0 PC Handler
Usage:
        ancpc [-f f_length -r rate -m mu -d delay -n -v]
        ancpc -l
        ancpc -s
        ancpc -e
        ancpc -t
        ancpc -h

 -f f_length : filter length, default 32
 -r rate     : sampling rate, default 10kHz
 -m mu       : adaptive constant, default 1e-20
 -d delay    : delay samples, default 0
 -n : reference option, receive reference input
 -l : load DSP program to the board
 -s : stop execution
 -e : reset to the board
 -t : dsp program test
 -v : verbose option, print details during execution
 -h : help and exit
```

Cuando el programa se ejecuta por primera vez, se debe usar la opción "ancpc -l"

Para hacer funcionar "anc.out", se debe escribir en la línea de comando "ancpc" junto con los valores de los parámetros del sistema. Por ejemplo, si se desea que el filtro tenga un tamaño de 32 coeficientes, la frecuencia de muestreo sea de 10 kHz y la constante de adaptación tenga el valor de 1e-19, entonces debe escribirse lo siguiente:

El código del programa completo aparece en el apéndice IV, solo aquí explicaré brevemente algunas secciones del código.

Estos son los encabezados que requiere el programa para compilar:

```
#include "c30coff.h"
#include "tms30.h"
#include "anc.h"
#include "..\getopt\getopt.h"
#include "..\dspanc\dspanc.h"
```

El encabezado "c30coff.h" fue creado por el Dr. Felipe Orduña que contiene rutinas para el control de la tarjeta. El "tms30.h" contiene las funciones que proporciona el fabricante de la tarjeta. El "getopt.h" es el encabezado de un programa que recibe las opciones que acompañan al invocar el programa "ancpc.exe" y las interpreta. "dspanc.h" contiene instrucciones que manipulan la tarjeta. "anc.h" contiene los valores iniciales de las variables en el programa.

## DSPANC.H

En el encabezado "dspanc.h" del apéndice VII se encuentran los identificadores de las rutinas de control de la tarjeta:

| | |
|---|---|
| int dspLoad( char *fname ); | Carga el programa definido por "*fname" a la memoria de la tarjeta |

Las siguientes funciones retornan un valor del mismo tipo definido por la función en la variable designada por *name:

```
long dspGet1Long( char *name, int *rcode );
float dspGet1Float( char *name, int *rcode );
int dspSet1Long( char *name, long value );
int dspSet1Float( char *name, float value );
```

Esta función cambia el valor de la variable "dspMessage" que se encuentra en la memoria de la tarjeta

```
int dspSendMessage( unsigned long msg );
```

## ANCPC.C→ANCPC.EXE

Definición de constantes usadas durante el programa:

| | | |
|---|---|---|
| #define OK | 0 | Mensaje de éxito |
| #define ERROR | 1 | Mensaje de error |
| #define C30_ADRESS | 0x0390 | Dirección de la tarjeta en la PC |
| #define C30_FILE | "anc.out" | Archivo ejecutable COFF en la tarjeta |

Conjunto de caracteres que definen la versión del programa:

```
#define  VERSION                "This is ANCPC program Version 1.0 PC Handler"
static char version[] = VERSION;
```

Conjunto de caracteres que despliegan la ayuda:

```
static char usage[] = VERSION "\n"
"Usage:\n"
"       ancpc [-f filter_length -r rate -m mu -d delay -n -v]\n"
"       ancpc -l\n"
"       ancpc -s\n"
"       ancpc -e\n"
"       ancpc -t\n"
"       ancpc -h\n\n"
" -f length  : filter length, default 32\n"
" -r rate    : sampling rate, default 10kHz\n"
" -m mu      : adaptive constant, default 1e-20\n"
" -d delay   : delay samples, default 0\n"
" -n : reference option, receive reference input\n"
" -l : load DSP program to the board\n"
" -s : stop execution\n"
" -e : reset to the board\n"
" -t : dsp program test\n"
" -v : verbose option, print details during execution\n"
" -h : help and exit\n";
```

Mensaje de error en la sintaxis de ejecución del programa:

```
static char bad_usage[] = "Error in command line. Try: ancpc -h";
```

Conjunto de opciones disponibles en la ejecución del programa:

```
static char optlist[] = "f:r:m:d:nlsetvh";
```

Declaración de variables globales:

```
int verbose = 0;                        Opción de visualización de parámetros durante
                                        la ejecución del programa
long id;                                Identificador del programa en la tarjeta
long dsp_state;                         Identificador del estado de la tarjeta
long f = Filter_length;                 Almacena el tamaño del filtro
long delay = DELAY;                     Almacena el número de señales de retardo
long ref = REF_OFF;                     Identificador de señal de referencia disponible
double mu = MU;                         Almacena la constante de adaptación
double rate = SAMPLE_RATE_HZ;           Almacena la frecuencia de muestreo de la tarjeta
double delaysec = DELAY;                Almacena el tiempo de retardo en segundos
```

El programa "ancpc.c" se divide en 5 funciones : dsp_load(), dsp_stop(), dsp_reset(), dsp_test() y dsp_run(). Todas estas funciones devuelven un valor entero que representará el éxito obtenido o un error en la ejecución del programa.

La función dsp_load() carga en la memoria de la tarjeta el código del programa ejecutable en la misma y verifica que se pueda comunicar la PC con la tarjeta.

```
int dsp_load(void)
{
 int result;                            Variable donde almacenará el resultado del proceso
 if (dspLoad(C30_FILE)) return(1);      Carga el archivo "anc.out" a la tarjeta, en caso de
                                        error devuelve "1"
 if (c30LoadSymbols(C30_FILE)) return(2);  Carga a la memoria de la PC las variables usadas en
                                        la tarjeta.
 result = dspSendMessage(DSP_READY);    Detecta si la tarjeta está lista para su ejecución
 if (result == 1) return(1);            En caso contrario devuelve un entero que
 if (result == 2) return(2);            representará un error

 c30ReleaseSymbols();                   Libera las variables antes cargadas a la memoria de
                                        la PC
```

```
return OK;                                  Si el programa logra llegar hasta aquí no se presentó
                                            algún error y devuelve un "0"
}
```

Esta función, dsp_reset(), realiza un proceso de RESET a la tarjeta:

```
int dsp_reset(void)
{
 if ((SelectBoard(C30_ADRESS)) != C30_ADRESS) return (1);
                                            Selecciona la dirección que se ha asignado a la
                                            tarjeta
 Reset();                                   Aplica un proceso de RESET a la tarjeta
 c30ReleaseSymbols();                       Libera de la memoria de la PC las variables usadas
                                            en la tarjeta
 remove("outdata");                         Elimina un archivo que contiene los valores actuales
                                                  de las variables de la tarjeta en caso de que
exista
  return OK;                                Regresa un cero en caso de haber ejecutado todos
                                            estos procesos con éxito
}
```

La función dsp_test_id() verifica que exista el programa "anc.out" actualmente en la memoria de la tarjeta.

```
int dsp_test_id(void)
{
 clock_t t;

 dspSendMessage(DSP_SET_ID);                Manda ejecutar la subrutina de fijar el identificador
                                            a "1" del programa "anc.out"
 dspSendMessage(DSP_READY);                 Pone en espera a la tarjeta
 t = clock() + 50;                          Espera un tiempo aproximado de 0.05 segundos
 while (clock() < t);
 if ((dspGet1Long("id", NULL)) != 1) goto fail_id;
                                            Si el identificador no se modificó salta a "fail_id"
 dspSendMessage(DSP_UNSET_ID);              Cambia el identficador a "1"
 dspSendMessage(DSP_READY);                 Pone en espera a la tarjeta
 t = clock() + 50;                          Espera nuevamente
 while (clock() < t);
 if ((dspGet1Long("id", NULL)) != 0) goto fail_id;  Si no se modificó el identificador salta a "fail_id"
 return OK;                                 Regresa un cero si el proceso se realizó con éxito
 fail_id : return(1);                       Regresa un uno indicando un error
 }
```

Rutina que evalúa el estado de la tarjeta y devuelve en el archivo "outdata" los valores que en ese momento se almacenan en las variables de la misma:

```
int dsp_test(void)
{
clock_t t;
FILE *fp;

if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return (1);
                                        Esta instrucción permite la elección de la tarjeta sin
                                              alterar su estado

if (c30LoadSymbols(C30_FILE)) return(2);
if (dsp_test_id() != OK) return(3);                    invoca la rutina de identificación del
                                                       programa
fp = fopen("outdata", "w");                            Abre un archivo físico de solo escritura
fprintf(fp, "%i\n", dspGet1Long("FI", NULL));          Obtiene los valores actuales de las
fprintf(fp, "%g\n", dspGet1Float("rate", NULL));       variables sin alterarlos y los escribe en
fprintf(fp, "%g\n", dspGet1Float("mu", NULL));         el archivo de memoria "fp" para
fprintf(fp, "%i\n", dspGet1Long("delay", NULL));       escribirlos a "outdata"
delaysec = dspGet1Long("delay", NULL) * (1 / dspGet1Float("rate", NULL));
fprintf(fp, "%g\n", delaysec);
fprintf(fp, "%i\n", dspGet1Long("ref", NULL));
fprintf(fp, "%i\n", dspGet1Long("state", NULL));
fclose (fp);                                           Cierra el archivo "outdata"
t = clock() + 500;
while (clock() < t);

c30ReleaseSymbols();

return OK;
}
```

La siguiente rutina ejecuta el programa con los valores actualizados en la línea de comando, si no se ha actualizado alguno, el programa asigna los valores definidos en "anc.h", además si se eligió la opción -v (verbose) en la línea de comandos, aparecerán en pantalla los valores que van tomando los parámetros de la tarjeta

```
int dsp_run(void)
{
 if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return (1);

 if (c30LoadSymbols(C30_FILE)) return (2);

 if (dsp_test_id() != OK) return(3);

 dspSendMessage(DSP_STOP);
 dspSendMessage(DSP_READY);
```

```
if (verbose) fprintf(stdout, "f = %i\n", f);
if (dspSetLong("Fl", f)) goto fail;
if (verbose) fprintf(stdout, "rate = %gkHz\n", rate/1000.0);
if (dspSetFloat("rate", (float) rate)) goto fail;
if (verbose) fprintf(stdout, "mu = %g\n", mu);
if (dspSetFloat("mu", (float) mu)) goto fail;
if (verbose) fprintf(stdout, "delay = %i\n", delay);
if (dspSetLong("delay", delay )) goto fail;
delaysec = (float) delay * (1 / rate);
if (verbose) fprintf(stdout, "delaysec = %g\n", delaysec);
if (verbose) fprintf(stdout, "reference = %i\n", ref);
if (dspSetLong("ref", ref)) goto fail;

dspSendMessage(DSP_INIT);
dspSendMessage(DSP_START);
dspSendMessage(DSP_READY);

c30ReleaseSymbols();

return OK;

fail: return (4);
}
```

La rutina dsp_stop detiene la ejecución del programa de filtrado sin descargar al programa de la memoria de la tarjeta:

```
int dsp_stop(void)
{
 if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return (1);

 if (c30LoadSymbols(C30_FILE)) return(2);

 if (dsp_test_id() != OK) return(3);

 if ((dspGetLong("state", NULL)) != DSP_ON) return(4);

 dspSendMessage(DSP_STOP);
 dspSendMessage(DSP_READY);

 c30ReleaseSymbols();

 return OK;
}
```

El programa principal al ser invocado en la línea del comando del DOS requiere algunos parámetros que serán interpretados por una función llamada "getopt()". Si el programa es invocado por una de las siguientes opciones: -l, -s, -e, -t, entonces realiza la función indicada, sin ejecutar la rutina dsp_run(), devolviendo a la consola de salida el resultado del proceso. Por otro

lado, si el programa es ejecutado usando algún otro parámetro, entonces la rutina a ejecutarse es dsp_run() desplegando en la pantalla el resultado del proceso.

```
int main(int argc, char *argv[])
{
 int opt;

 while ((opt = getopt(argc, argv, optlist)) != EOF)
 {
  switch (opt)
  {
   case 'f': f = atoi(optarg);break;
   case 'r': rate = atof(optarg); break;
   case 'm': mu = atof(optarg); break;
   case 'd': delay = atoi(optarg); break;
   case 'n': ref = REF_ON;break;
   case 'l': fprintf(stdout, "%i\n",dsp_load());exit(EXIT_SUCCESS);
   case 's': fprintf(stdout, "%i\n",dsp_stop()); exit(EXIT_SUCCESS);
   case 'e': fprintf(stdout, "%i\n",dsp_reset());exit(EXIT_SUCCESS);
   case 't': fprintf(stdout, "%i\n",dsp_test());exit(EXIT_SUCCESS);
   case 'v': verbose = 1; break;
   case 'h': puts(usage); exit(EXIT_SUCCESS);
   default: puts(bad_usage); exit(EXIT_FAILURE);
  }
 }
 fprintf(stdout,"%i\n",dsp_run());

 return OK;
}
```

Como ya he venido mencionando, al ejecutar las funciones de "ancpc.c", se devuelve un valor entero a la pantalla de la PC indicando el resultado del proceso. Si el valor regresado es un cero, el programa ejecutó con éxito la tarea indicada. Por otro lado, si el valor es diferente de cero y dependiendo de la función que se haya querido ejecutar, es posible interpretar este valor con el siguiente código de errores:

función **dsp_load() ó (línea de comando: ancpc -l)** :

1. No existe en el directorio donde se encuentra "ancpc.exe" el archivo ejecutable COFF "anc.out" o la dirección asignada a la tarjeta no es 0x390.
2. No se cargó el nombre de las variables de la tarjeta pertenecientes a "anc.out" a la memoria de la PC.
3. No se puede establecer comunicación entre la tarjeta y la PC.

función **dsp_reset() ó (línea de comando: ancpc -e)** :

1. La dirección de la tarjeta no corresponde a 0x390.

función **dsp_test ó (línea de comando: ancpc -t)** :

1. La dirección asignada a la tarjeta no corresponde a 0x390.
2. No se cargó el nombre de las variables de la tarjeta pertenecientes a "anc.out" a la memoria de la PC.
3. El programa "anc.out" no se encuentra cargado a la tarjeta.

función **dsp_run ó (línea de comando: ancpc [-f filter_length -r rate -m mu -d delay -n -v])** :

1. La dirección asignada a la tarjeta no corresponde a 0x390.
2. No se cargo el nombre de las variables disponibles para "anc.out" en la tarjeta a la memoria de la PC.
3. El programa "anc.out" no se encuentra cargado a la tarjeta.
4. No se pueden escribir datos a la tarjeta.

función **dsp_stop ó (línea de comando: ancpc -s)** :

1. La dirección asignada a la tarjeta no corresponde a 0x390.
2. No se cargó el nombre de las variables de la tarjeta pertenecientes a "anc.out" a la memoria de la PC.
3. El programa "anc.out" no se encuentra cargado a la tarjeta.
4. El programa "anc.out" no está en ejecución.

### 7.3.3 PROGRAMA GRÁFICO EN AMBIENTE WINDOWS©

El programa creado para ambiente Windows©, "anc.exe", es un programa que ejecuta a "ancpc.exe" mediante una línea de comando del DOS. El código escrito para este programa no se explica aquí puesto que no resulta importante para el sistema ya que solo es una interfaz gráfica que proporciona una mejor presentación al usuario. Además, si se intentara explicar el código se necesitaría tener conocimientos previos sobre lenguajes visuales y programación orientada a objetos. Si el lector está interesado en el diseño de este programa, en los apéndices del X al XIII, se muestran las partes más significativas del código. En la fig. 7.8 se muestra la ventana del programa principal de "ANC FILTER".

Este menú principal contiene dos submenús mostrados en la fig. 7.9 y 7.10. Al seleccionar el submenú de **Execute** aparecerán las opciones de éste que son: **Execute** y **Exit ANC....** La opción **Execute** abrirá un diálogo que establece la "comunicación" entre el programa para Windows© y la tarjeta mediante el programa "ancpc.exe", es decir, el diálogo enviará una línea de comandos con los parámetros que se encuentran en este y se ejecutará con "ancpc [lectura de parámetros del diálogo]". Este diálogo se muestra en la fig. 7.11. La opción de **Exit ANC...** abandona la ejecución del programa "ANC FILTER" sin alterar el estado de la tarjeta, es decir, si esta se encuentra activa, permanecerá así con los valores de los parámetros definidos en ese momento.

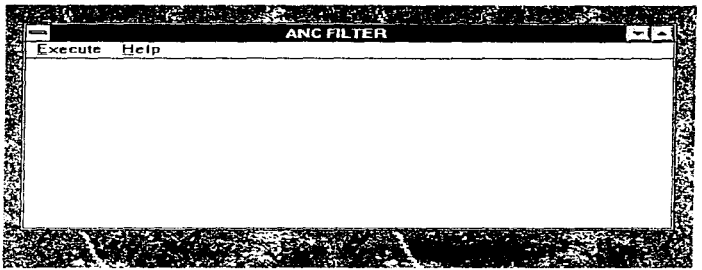



Fig. 7.8 Ventana principal del programa ANC FILTER para ambiente Windows©.

En submenú **Help**, aparecerán dos opciones que son **Help** y **About ANC...**. Al elegir la opción **Help**, se abrirá una ventana de ayuda al usuario en ambiente Windows©, similar a la ayuda de cualquier programa para Windows©. La opción de **About ANC...**, presenta la versión del programa, fecha de creación y el autor.

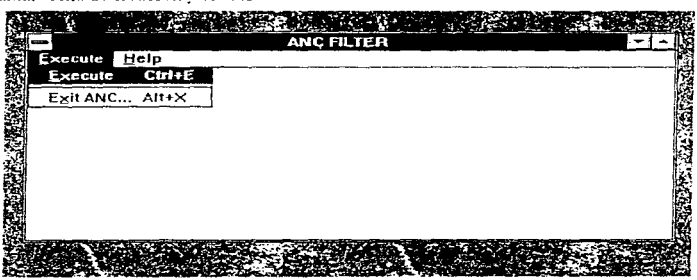



Fig. 7.9 Submenú de Execute del menú principal de ANC FILTER.

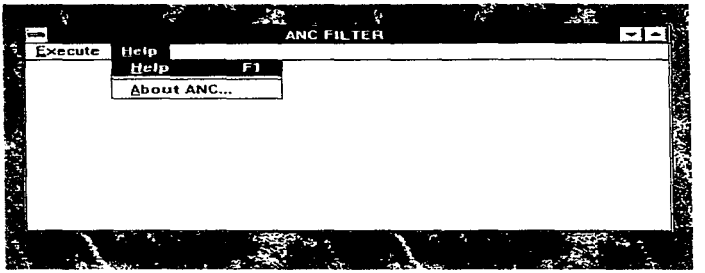



Fig. 7.10 Submenú de Help del programa ANC FILTER.

La figura 7.11 muestra el diálogo principal que establece la comunicación con la tarjeta, si el programa se ejecuta por primera vez, el diálogo aparecerá con los valores por *default* que han sido declarados previamente en "anc.h". Si el programa ha sido previamente ejecutado por Windows© o desde el *prompt* de DOS, al abrir nuevamente este diálogo, el programa mandará a la línea de comandos la opción de *test* que devolverá los valores de los parámetros que contiene en ese momento el filtro y así la ventana del diálogo será actualizada.

Los botones disponibles en esta ventana son: **Execute, Load/Disload, Stop, Close** y **Refresh**. Si el programa "anc.out" no se encuentra en la memoria de la tarjeta, el único botón que estará a disposición será **Load** y las demás opciones estarán deshabilitadas. Una vez presionado este botón (el programa "anc.out" estará entonces en la memoria de la tarjeta) cambiará con el nombre de **Disload**, mismo que descargará a "anc.out". También estarán disponibles los botones de **Execute** y **Refresh**. El botón de **Execute** hará funcionar el proceso de filtrado con los parámetros que aparecerán en ese momento en la ventana de diálogo, el botón de **Stop** cambiará su estado para poder detener la ejecución del filtro. El botón de **Refresh** actualizará la ventana de diálogo tomándolos desde la tarjeta. El botón de **Close** cierra la ventana de diálogo sin alterar el estado de la tarjeta ni sus parámetros.

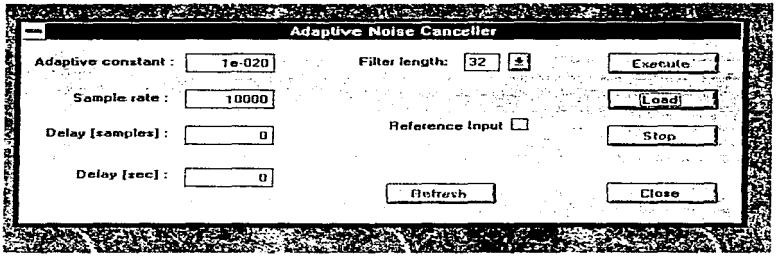



Fig. 7.11 Ventana de dialogo para comunicación con la PC.

La opción de **Reference Input**, indica al programa "anc.out" si se dispone de una señal de referencia o no.

Los parámetros del filtro, son: **Adaptive constant**, **Sample rate**, **Delay[samples]**, **Delay[sec]** y **Filter length**. Estos parámetros pueden ser modificados excepto **Delay[sec]**, ya que en este aparecerá el resultado del cálculo del tiempo producido por el retardo indicado en **Delay[samples]**. Es importante tener algunas consideraciones al ajustar estos parámetros debido a la dependencia que existe entre estos.

La asignación de estos indicadores son: **Adaptive constant** que se refiere al valor que toma la constante de adaptación en el algoritmo LMS, *vid. infra* p. 45. **Sample rate**, es la frecuencia de muestreo con la que los convertidores trabajan. **Delay[samples]** es el número de muestras que serán atrasadas para obtener una señal que sirva de referencia al filtro. **Filter length** indicará el tamaño del filtro (número de coeficientes) adaptivo, este parámetro debe tomar un valor que sea potencia de dos ($2^n$), ya que se requiere un valor así para realizar adecuadamente la suma de convolucion en el programa "anc.out". Se pueden elegir entre 8, 16, 32, 64, 128 y 256. Debido a que el vector que recibe las muestras de entrada al filtro tiene un tamaño de 256 localidades, el número máximo de muestras de retardo es 255.

El programa logra alcanzar aproximadamente una frecuencia de muestreo de 25 kHz cuando el tamaño del filtro es de 32 coeficientes. Si se excede este límite, el programa puede causar disturbios en su funcionamiento. Es importante aclarar que el programa para Windows© no es el soporte del programa en la tarjeta, "anc.out", es decir, si no se dispusiera "ANC Filter" o llegara a fallar el programa "anc.out" puede controlar mediante el programa "anepc.exe" desde el *prompt* del sistema operativo.

## 7.4. RESULTADOS DEL SISTEMA EN TIEMPO REAL

En esta sección presentaré los resultados del funcionamiento del sistema en tiempo real. Las mediciones del filtro se llevaron a cabo con un analizador de espectros digital capturando los resultados en archivos con formato ASCII que posteriormente fueron graficados por Matlab©. En la fig. 7.12 se muestra el diagrama de conexiones del equipo de audio y medición que se utilizó para realizar las pruebas. Como el filtro funciona tanto cuando se dispone de alguna señal de referencia como cuando no existe ésta, las pruebas incluyen estos dos casos. Las señales de referencia usadas son la misma que se usaron para interferir a la señal original. Las señales que se consideraron de interés son voz y musicales interferidas por diversos tipos de ruidos (senoidal, cuadrada, producida por un torno y ruido blanco). Finalmente se aplicó el sistema a una grabación que contiene una conferencia interferida por un ruido producido quizá por una conexión de tierra inestable.

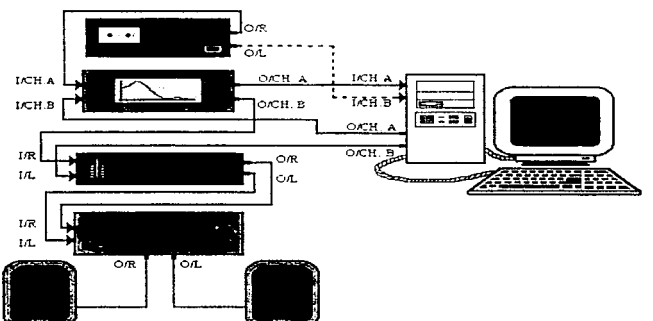



Fig. 7.12 Conexión del equipo de audio y medición para la realización de las pruebas.

Además de observar los resultados mediante el analizador de espectros, también fue posible escuchar las señales de entrada y salida al sistema a través del equipo de audio conectado a la salida de la tarjeta, como se observa en la fig. 7.12, con el fin de verificar si se logra alcanzar la inteligibilidad de la señal de interés propuesta al inicio de la tesis. Como el programa "anc.out" envía las señales "e" (señal de error del sistema) y "y" (señal de salida del filtro adaptivo), *vid. infra* p. 77, a través de los DAC, es posible escuchar estas señales en forma separada. Así, el equipo de audio reproduce por un canal del amplificador la señal de interés "recuperada" y por otro, la señal "extraída" que representa al ruido.

Antes de continuar, debo mencionar que los valores de los parámetros del sistema usados en estas pruebas, fueron los que lograron el mejor desempeño del sistema. Al intentar con otros

valores, algunos de ellos producían resultados similares a los que se presentan aquí y otros no satisfacían el fin de esta tesis.

### 7.4.1 INTERFERENCIA CON UNA SEÑAL SENOIDAL.

La primera parte de las mediciones consistió en interferir una señal musical con una señal senoidal de 600 Hz con el fin de observar el comportamiento del sistema para una señal periódica "pura". Los parámetros del filtro y la frecuencia de corte del filtro pasa-bajas externo al sistema se muestra en la siguiente tabla:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.13a | 19 | 32 | 1e-20 | present | 10 |
| 7.13b | 19 | 32 | 1e-20 | 92 | 10 |

Los valores de estos parámetros fueron los valores óptimos con los que se produjo el mejor resultado del sistema. En la fig. 7.13a se muestra el espectro del resultado cuando se considera que se dispone de una señal de referencia. De ésta se observa claramente que la señal de entrada (línea continua) presenta un "pico" justo en la frecuencia de la señal de interferencia (600 Hz) mientras que el espectro de la salida del sistema (línea discontinua) se nota como es atenuada la potencia de la señal precisamente a esta frecuencia, también se puede ver en esta figura que se pierde un poco la potencia de la señal después de los 3000 Hz, por lo que se puede inferir que el sistema se comporta además como un filtro pasa-bajas.

Por otro lado, en la fig. 7.13b se observan los espectros de la señal de entrada (línea continua) y salida (línea discontinua) al sistema cuando se considera que no se dispone de una señal de referencia externa. Así pues, se debe producir un retraso de la señal de entrada para tener una señal de referencia. El tiempo de retardo (muestras de retardo) de la señal de entrada para este tipo de interferencia no resultó ser de gran importancia, pues al aplicar diferentes tiempos, los resultados fueron satisfactorios. Quizá esto se debe a que la señal de ruido es una señal periódica "pura".

Es notable la diferencia entre los espectros de la fig. 7.13a y 7.13b, por un lado, en la fig. 7.13a se nota que no se pierde demasiado la señal original, y por otro, en la fig. 7.13b se nota cierta pérdida de la señal original provocada por el sistema. Al escuchar estos resultados mediante el equipo de audio, no se notan demasiado estas perdidas y se logra el objetivo propuesto originalmente que es la inteligibilidad de las señales.

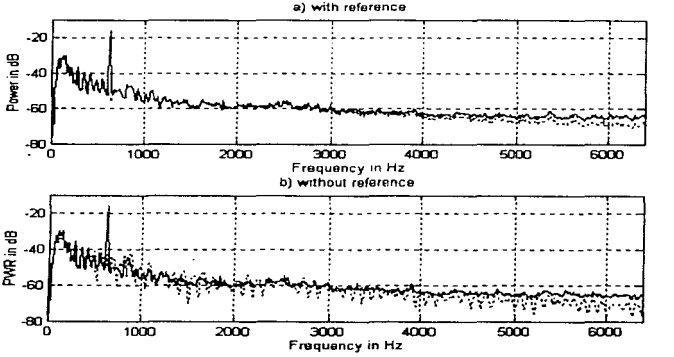



Fig. 7.13 Espectros de la entrada, línea continua y salida, línea discontinua, del sistema para la prueba con una señal de interferencia senoidal y señal de interés musical:
a) con señal de referencia; b) sin señal de referencia.

### 7.4.2 INTERFERENCIA CON UNA SEÑAL CUADRADA.

Para el caso anterior, se puede afirmar que se logra una buena cancelación de la señal de ruido inclusive cuando no se dispone de una señal de referencia, sin embargo, cuando la señal de interferencia es un poco más compleja, el comportamiento del sistema no es tan satisfactorio. Una señal cuadrada se puede representar mediante una sumatoria de senos y cosenos con una frecuencia múltiplo de la frecuencia de la fundamental. Así pues, el espectro es una señal un tanto compleja en sentido de que no solo se observa un pico como en el caso de la señal senoidal, sino también una serie de armónicos. En esta prueba se usó una señal cuadrada pasada por un filtro pasa-bajas con frecuencia de corte de 3.15 kHz. Las pruebas para este tipo de señal de ruido recaen en tres casos: (1) señal de voz con una señal cuadrada de 60 Hz como interferencia. (2) señal musical con una señal cuadrada de 60 Hz como interferencia y (3) señal musical con una señal cuadrada de 120 Hz como interferencia.

Caso 1: Los espectros se presentan en la fig. 7.14a cuando se dispone la señal de referencia, y la fig. 7.14b muestra los espectros de los resultados sin considerar la referencia externa. Nuevamente es notable la diferencia para los dos casos, cuando se cuenta con la referencia casi se logra la cancelación total de la señal que interfiere y cuando no se considera la señal de referencia, no se logra esta cancelación de una manera satisfactoria.

Los parámetros usados en esta prueba son:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.14a | 6.5 | 128 | 1e-19 | present | 3.15 |
| 7.14b | 10 | 32 | 1e-20 | 93 | 3.15 |

Los resultados producidos por el sistema se logran apreciar de mejor forma al escucharlos mediante el equipo de audio. Originalmente al escuchar la grabación que entra al sistema (señal de voz mezclada con la señal cuadrada), solo se escucha un zumbido bastante molesto sin lograr siquiera distinguir la voz. La salida del sistema reproduce por un canal de la tarjeta la señal de voz con un leve sonido que no es tan molesto como el original y por otro canal solo la señal de interferencia. En el espectro de las figuras se nota claramente que la señal de interés (voz) no se recupera en su totalidad, pues algunos componentes de la señal de interferencia permanecen aún después de ser filtrada.

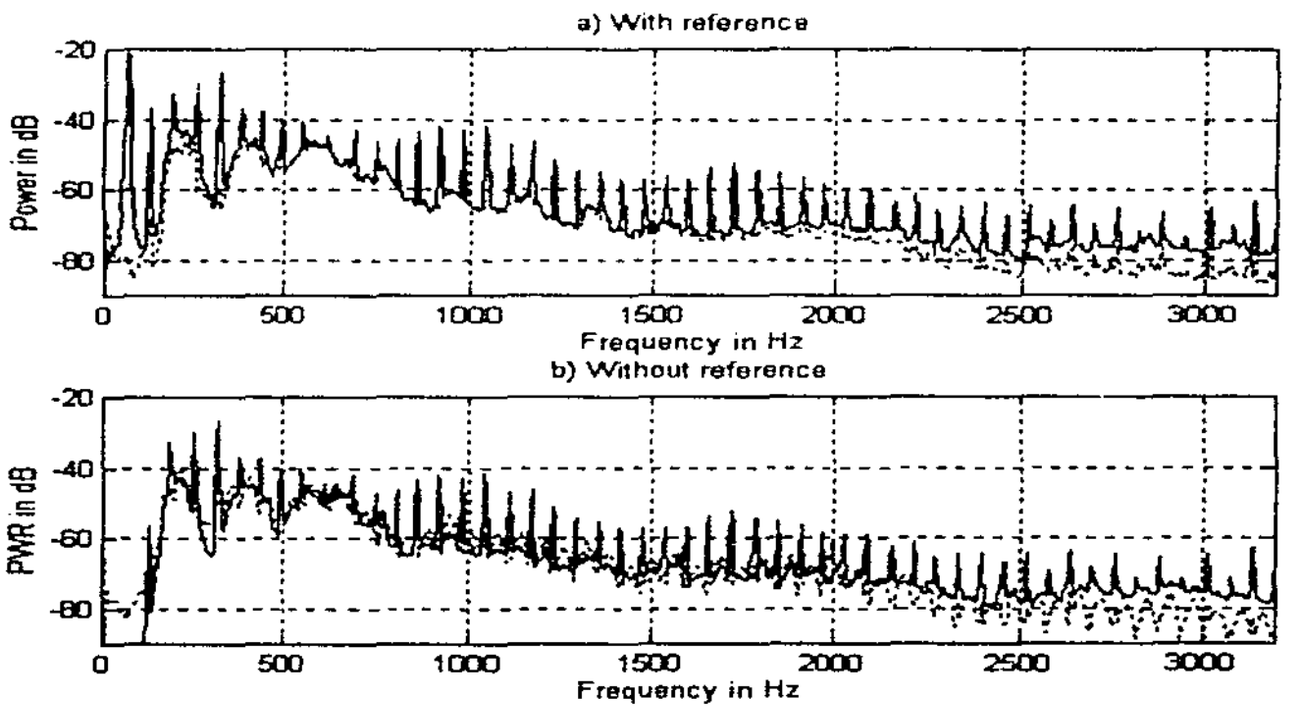



**Fig. 7.14 Espectros de los resultados de la prueba de señal de voz interferida con una señal cuadrada a 60 Hz, entrada, línea continua y salida, línea discontinua: a) con referencia y b) sin referencia.**

Caso 2: Para esta prueba se usa la misma señal cuadrada de 60 Hz. como interferencia y como señal de interés una señal musical. Los espectros para este caso se presentan en la fig. 7.15a cuando se considera la señal de referencia externa y 7.15b cuando no se considera esta.

La tabla de parámetros del filtro son:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.15a | 19 | 32 | 1e-19 | present | 8 |
| 7.15b | 19 | 32 | 1e-20 | 132 | 8 |

Los datos de la tabla anterior fueron los que lograron obtener el mejor resultado, pues se intentaron varios valores para los parámetros. Los resultados no son favorables puesto que no se logra una cancelación como en los casos anteriores. La señal cuadrada domina a la señal musical, es decir, solo se escucha ligeramente la señal musical detrás del molesto sonido producido por la señal cuadrada y al escuchar los resultados en el equipo de audio, se logra escuchar la música aunque no con muy buena claridad. Los espectros para esta prueba representan mejor este resultado, se puede ver que en ambos no se reduce lo suficiente la señal de ruido de la señal de entrada.

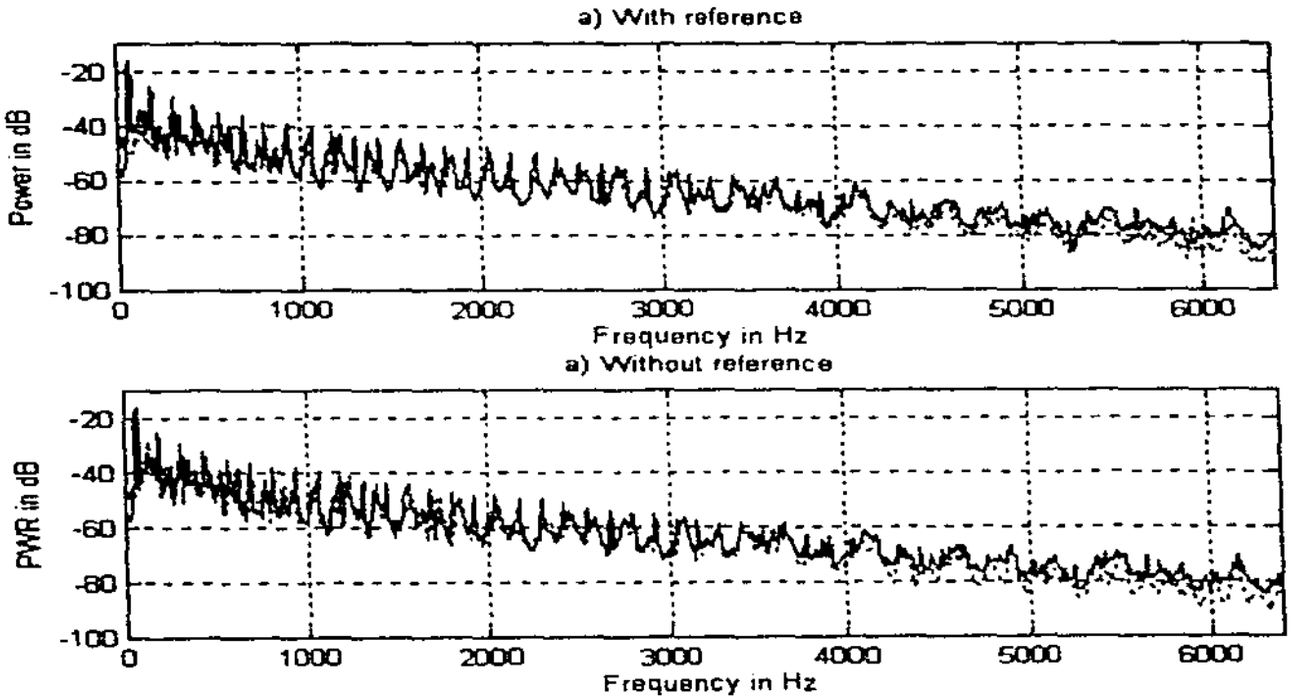



Fig. 7.15 Espectros de los resultados de la prueba de la señal musical interferida con una señal cuadrada a 60 Hz, entrada, línea continua y salida, línea discontinua:
a) con referencia y b) sin referencia

Caso 3: Ahora bien, usando una señal cuadrada como interferencia pero variando la frecuencia de ésta a 120 Hz. se logró lo siguiente: el espectro de la fig. 7.16a representa la entrada (línea continua) y salida (línea discontinua) al sistema cuando se dispone de la señal de referencia.

Los parámetros del filtro son:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.16a | 19 | 32 | 1e-20 | present | 8 |
| 7.16b | 19 | 32 | 1e-20 | 92 | 8 |

En la fig. 7.16b se presentan los espectros de las señales cuando no se considera la señal de referencia.

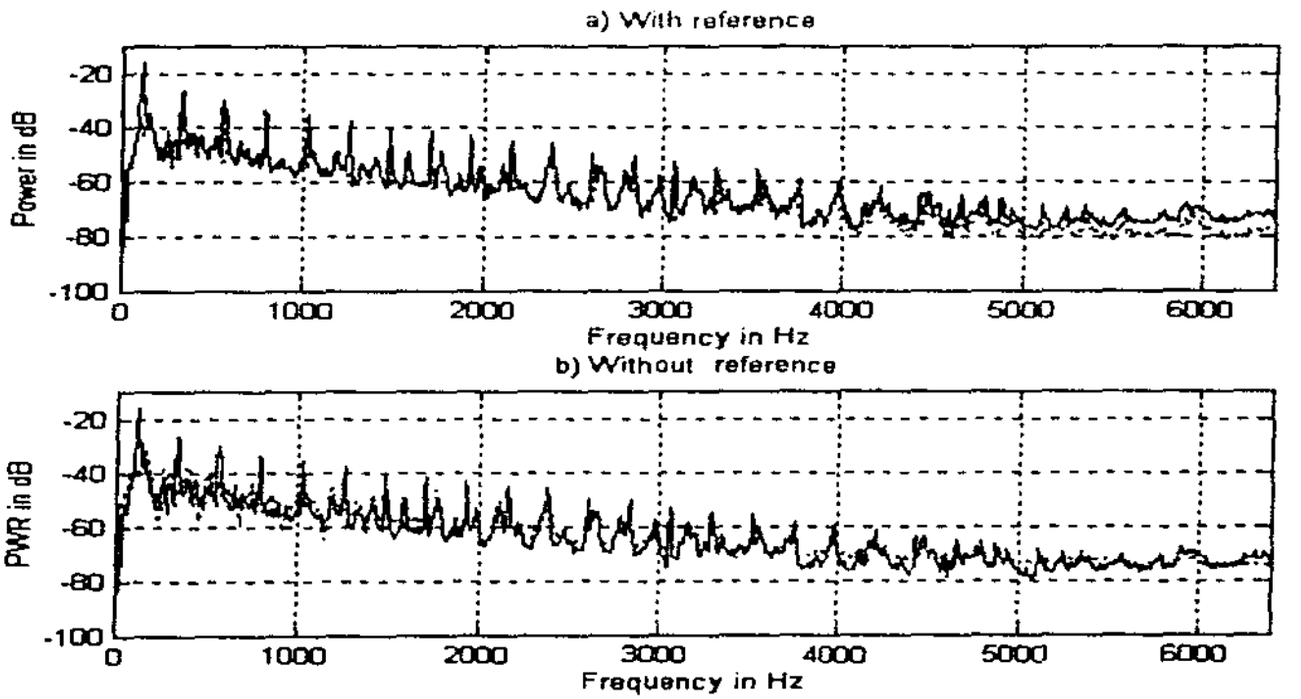



Fig. 7.16 Espectros de los resultados de la prueba con una señal musical interferida por una señal cuadrada a 120 Hz. Entrada al sistema, línea continua y salida del sistema línea discontinua: a) con referencia y b) sin referencia.

Ambos resultados son bastante similares y es claro que no se logró el objetivo de cancelar la señal de ruido. Con respecto a los resultados audibles, como en el caso anterior, solo se logro distinguir la señal de interés (musical).

### 7.4.3 INTERFERENCIA CON UN RUIDO PRODUCIDO POR UN TORNO

Para esta prueba se usa como señal de interferencia un ruido producido por un torno aplicado a una señal de voz. Los resultados obtenidos fueron bastante favorables, pues se logró eliminar casi en su totalidad la interferencia. Solo fue posible esto usando la misma interferencia como señal de referencia, pues aunque el ruido es un tanto periódico, su espectro parece un señal de ruido blanco. El espectro de esta prueba se observa en la fig. 7.17.

Esta prueba la intenté realizar tomando como señal de interés una señal musical, sin embargo no fue posible eliminar de ninguna manera la interferencia, es decir, considerando la señal de referencia y sin esta.

Los parámetros del filtro son:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.17 | 6.5 | 128 | 1e-20 | present | 3.15 |

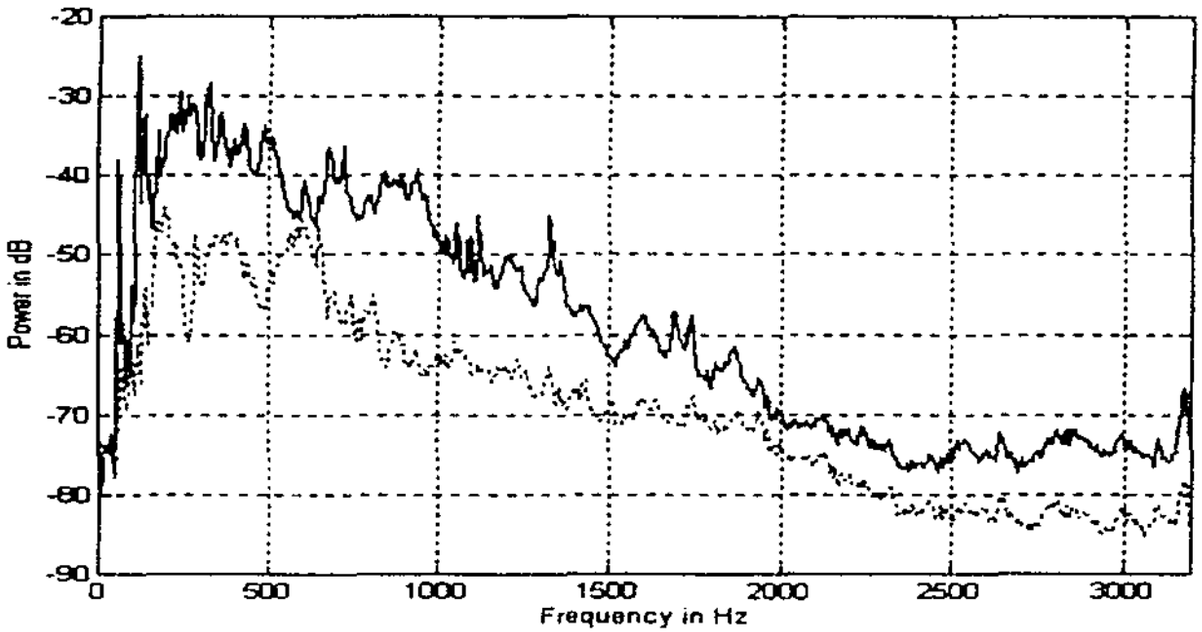



Fig. 7.17 Espectros del resultado de la prueba con señal de voz interferida con una señal producida por un torno. Entrada al sistema, línea continua y salida, línea discontinua.

### 7.4.4 INTERFERENCIA CON UNA SEÑAL DE RUIDO BLANCO

En este caso se usó como interferencia una señal de ruido blanco de 20 kHz pasada por un filtro pasa-bajas a 3.15 kHz. Las señales de interés son voz y musical. Los resultados se observan en los espectros de las fig. 7.18a (para el caso de señal de voz) y 7.18b (para el caso de señal musical).

Los parámetros del filtro son:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ Delay [samples] | low pass filter [kHz] |
|---|---|---|---|---|---|
| 7.18a | 19 | 32 | 1e-20 | present | 8 |
| 7.18b | 19 | 32 | 1e-20 | 92 | 8 |

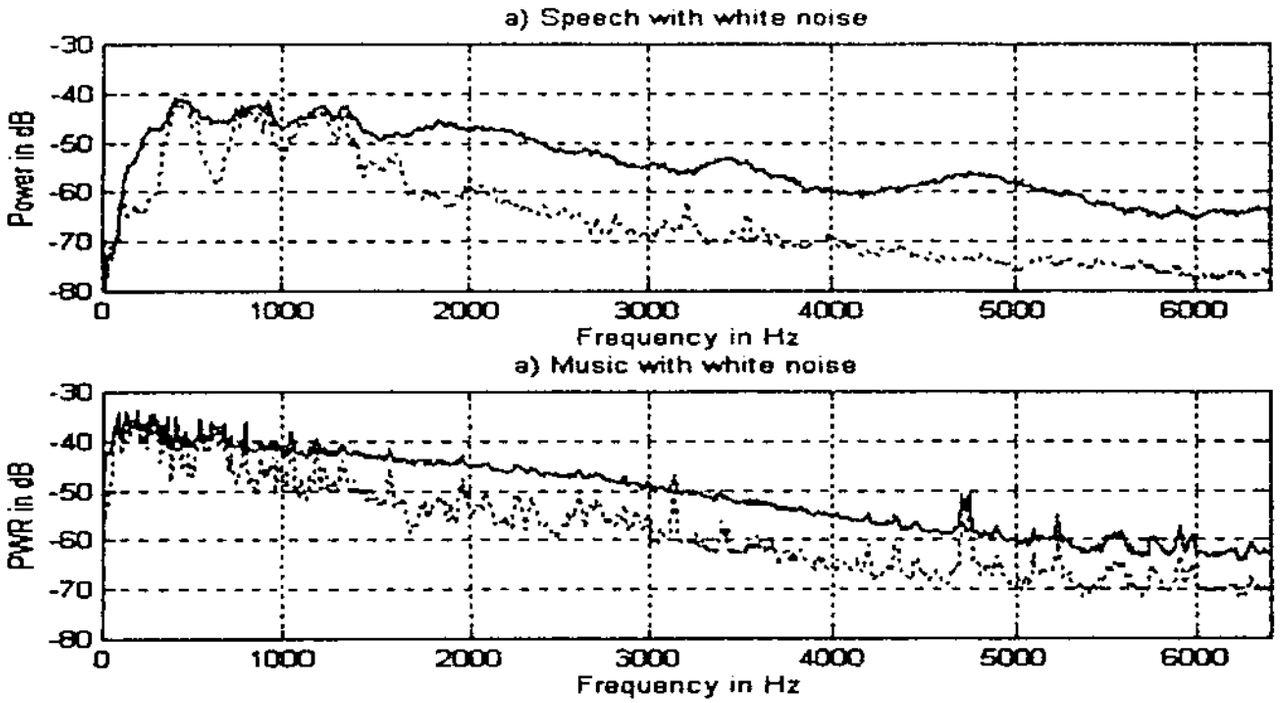



Fig. 7.18 Espectros de los resultados obtenidos del sistema usando ruido blanco de 20 kHz como interferencia. Entrada al sistema, línea continua y salida del sistema línea discontinua usando como señal de interés: a) voz y b) música

Estos resultados son bastante buenos, pero al aplicar el sistema para le caso que no se considera la señal de referencia, debido a la no periodicidad de esta señal, no fue posible la cancelación del ruido.

### 7.4.5 APLICACIÓN DEL SISTEMA A UN CASO COMÚN DE GRABACIÓN DE AUDIO

Finalmente, la última de las pruebas realizadas fue para un problema en el que se tiene una grabación de una conferencia interferida por un ruido periódico. Considero que esta señal de interferencia proviene de la línea eléctrica cuya frecuencia es de 60 Hz y ocasionado quizá por una conexión inestable de tierra. La señal producida por esta falla es muy parecida a una señal cuadrada. Esta grabación fue enviada a la sección de acústica del Centro de Instrumentos de la UNAM para una posible solución. Fue aquí donde surgió la idea de crear un cancelador de interferencia adaptiva y se propuso como tema de tesis. Los parámetros usados en el sistema son los siguientes:

| Fig. | Sample rate [kHz] | Filter Length | Adaptive Constant | Reference Input/ delay [muestras] | band pass filter [kHz] |
|---|---|---|---|---|---|
| 7.19 | 19 | 32 | 1e-20 | 100 | 0.15 - 3.15 |

Como este caso constituye un ejemplo de un problema muy común, lo analizaré más detalladamente que los demás. La fig. 7.19a muestra el espectro de la señal original después de pasarla por un filtro pasa-bandas externo al sistema. Los "picos" de mayor potencia que se observan en ésta figura corresponden a los armónicos de una señal periódica. La potencia del "pico" más alto tiene un valor de -20.2 dB correspondiente a 184 ± 4 Hz. El rango de error en la medición es debido al analizador de espectros digital usado en las mediciones. En la fig. 7.19b se tiene la gráfica del espectro de la señal después de aplicar el sistema de cancelación. La potencia de la señal a la frecuencia de 184 Hz, es -49.3 dB, esto significa que la señal de entrada se atenuó 29.1 dB en esa frecuencia. Si se toman las diferencias para todas las frecuencias, se obtiene el espectro de la fig. 7.20a y el promedio de estas diferencias, resulta ser de 19.82 dB, considerando solo la atenuación en las frecuencias en que se presentan los "picos" más altos del espectro de 7.19b. Esta cantidad, representa de alguna forma una medida del funcionamiento del filtro.

Ahora bien, el analizador de espectros es capaz de calcular la relación de la salida con la entrada que representa la función de transferencia del filtro. La fig. 7.20b representa ésta relación en función de la frecuencia. Así pues, la respuesta en frecuencia de este filtro adaptivo tiene un comportamiento similar al de un filtro peine (comb-filter). Los filtros peine pueden verse como una serie de filtros de muesca (notch-filter) en el cual los nulos ocurren periódicamente a través de la banda de frecuencia. Estos nulos corresponden a frecuencias múltiplo de una frecuencia establecida y la función de transferencia de un filtro de estos contiene D ceros sobre el círculo unitario, donde D es el número de muestras de retardo en un filtro FIR.

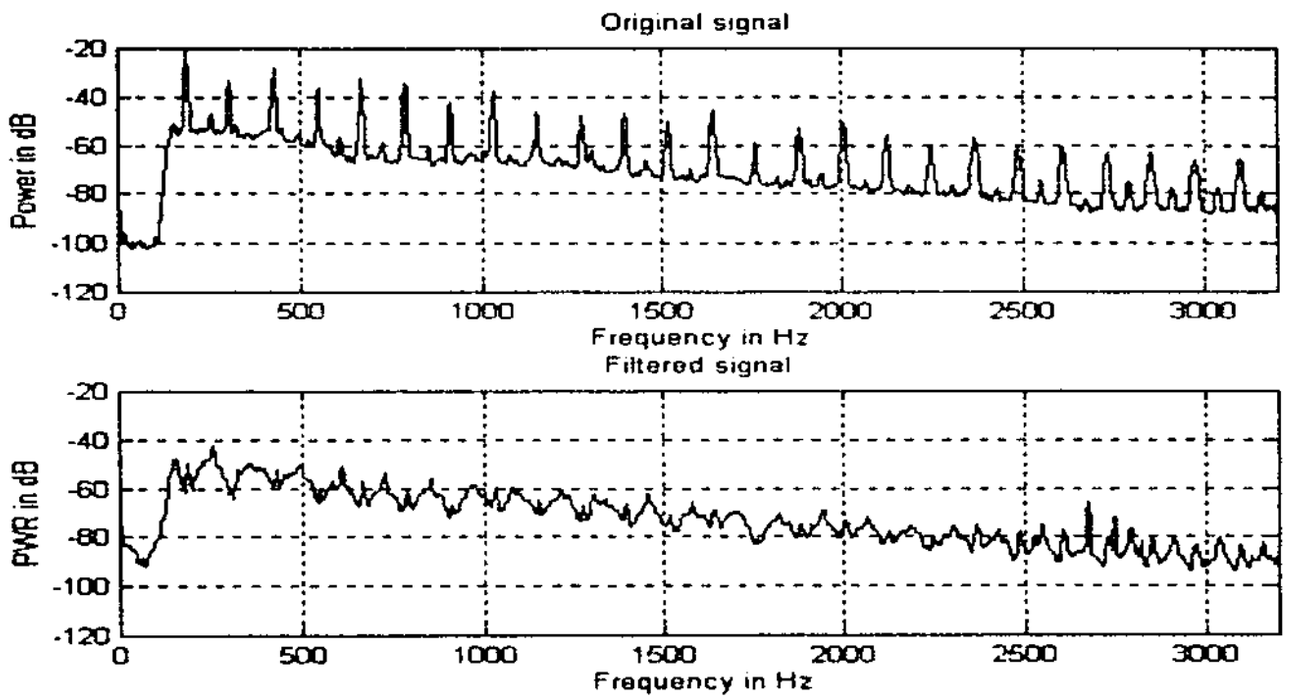



Fig. 7.19 Espectros de la a) señal original y b) señal filtrada.

De esta forma, el número de retardos del filtro adaptivo y la frecuencia de muestreo usada estarán directamente relacionados con el periodo de la señal que se quiere cancelar. Es importante hacer hincapié que para este tipo de problemas sólo es posible una solución aplicando un retardo a la señal de entrada primaria y así tener una versión correlacionada del ruido y a su vez decorrelacionada con la señal de interés. Como la señal fue muestreada a 19 kHz y el retardo empleado en el proceso fue de 100 muestras, esto equivale aproximadamente a 5.2 ms de retraso. Por otro lado, el periodo de la señal de interferencia (184 ± 4 Hz) es de aproximadamente 5.4 ± 1 ms. Esto significa que la señal de entrada al filtro se retrasó el tiempo equivalente al periodo de la frecuencia más baja en la señal de interferencia.

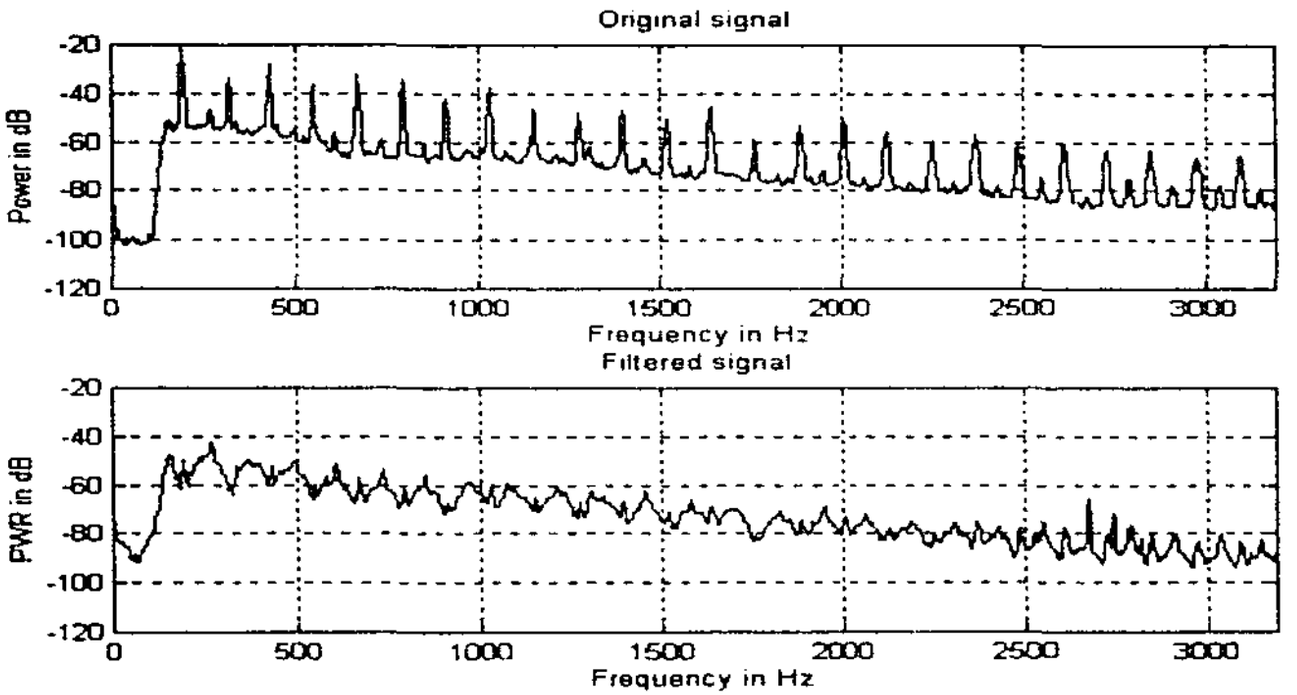



Fig. 7.19 Espectros de la a) señal original y b) señal filtrada.

De esta forma, el número de retardos del filtro adaptivo y la frecuencia de muestreo usada estarán directamente relacionados con el periodo de la señal que se quiere cancelar. Es importante hacer hincapié que para este tipo de problemas sólo es posible una solución aplicando un retardo a la señal de entrada primaria y así tener una versión correlacionada del ruido y a su vez decorrelacionada con la señal de interés. Como la señal fue muestreada a 19 kHz y el retardo empleado en el proceso fue de 100 muestras, esto equivale aproximadamente a 5.2 ms de retraso. Por otro lado, el periodo de la señal de interferencia (184 ± 4 Hz) es de aproximadamente 5.4 ± 1 ms. Esto significa que la señal de entrada al filtro se retrasó el tiempo equivalente al periodo de la frecuencia más baja en la señal de interferencia.

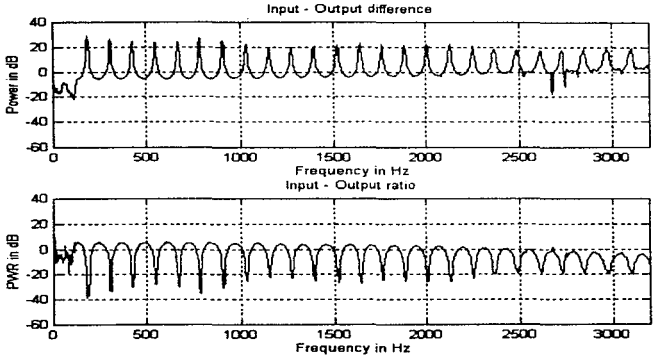



Fig. 7.20 Espectros de: a) relación salida/entrada del sistema calculada en Matlab© y b) relación salida/entrada obtenida del analizador de espectros.

# Capítulo 8

## CONCLUSIONES

En los capítulos 2 y 3 se presentó una breve introducción a la teoría de sistemas y señales, así como algunos conceptos sobre predicción y filtrado óptimo. En el capítulo 3, además, se expone el criterio de minimización MMSE para el problema de predicción. En el capítulo 4 se mencionan algunos algoritmos adaptivos y principalmente el LMS que es usado para el desarrollo del sistema creado en esta tesis. Este algoritmo resulta fácil de implementar y no requiere un gran número de operaciones para actualizar los coeficientes del filtro. En el capítulo 5 se dio una introducción a los sistemas adaptivos y las aplicaciones más comunes de estos sistemas. El capítulo 6 describe en forma general el DSP TMS320C30 de Texas Instruments que se uso como elemento de *hardware* para la implementación del sistema en tiempo real y también se dio una descripción del *software* necesario para el control del DSP y desarrollo de programas que funcionan en él. En el capítulo 7 se presenta el desarrollo del sistema de cancelación de interferencia adaptivo, también se explica el funcionamiento de los programas y como pueden ejecutarse. En este capítulo, se muestran también los resultados obtenidos en la evaluación del sistema analizando con mayor detenimiento una aplicación común.

Como se expuso en el capítulo 5, existen diversas aplicaciones de los sistemas adaptivos que pueden fácilmente implementarse en un DSP; sin embargo, impulsado por el interés que tengo en el área de procesamiento digital de señales de audio y la necesidad de desarrollar sistemas de cancelación de interferencias - aunque el objetivo en esta tesis solo fue lograr una buena inteligibilidad de las señales de audio - tomé la decisión de desarrollar esta aplicación en particular. El algoritmo adaptivo mas fácil de implementar es el LMS y para esta aplicación produjo muy buenos resultados. La programación de este algoritmo en el DSP, no requirió grandes operaciones aritméticas ni tampoco adaptaciones de subrutinas para su codificación. El sistema logró satisfactoriamente la cancelación de la mayoría de las señales que se usaron como prueba. Fue posible cancelar las señales que por su naturaleza son periódicas "puras", para la señal cuadrada que es una señal igualmente periódica, pero más rica en frecuencia, se logró una cancelación parcial de ésta. Cuando se tiene como señal de interferencia aleatoria como es el caso

del ruido blanco, se alcanzó una buena inteligibilidad de las señales de interés puesto que la señal de ruido blanco es una señal estacionaria, sin embargo, solo se logró esta cancelación tomando como señal de referencia la misma fuente de ruido. En la práctica cotidiana, las aplicaciones de mayor importancia son aquellas en las que no se dispone de una señal de referencia externa y no se conocen las características de esta. Para intentar lograr la cancelación de la señal de interferencia, fue necesario aplicar un retraso a la señal de entrada al sistema para así generar la señal de referencia. El resultado que se obtuvo al aplicar el sistema resultó bastante favorable y para los fines específicos de esta tesis - inteligibilidad de la conversación - se tuvo una perfecta inteligibilidad. Es posible afirmar que se logró un buen sistema de cancelación de interferencia. Por desgracia, en esta última aplicación no se conocían las características de la señal original, por lo que, no fue posible una caracterización más exacta del funcionamiento del sistema.

Aunque como afirme anteriormente, los resultados fueron satisfactorios, estos se podrían superar usando un DSP con mejores cualidades en cuanto a capacidad de almacenamiento, rapidez, etc., sin menospreciar las de la tarjeta usada.

# APÉNDICES

## APÉNDICE I

## MATRIZ POSITIVA DEFINIDA

Una matriz Hermitiana **A** es positiva definida, si para todos los vectores $\mathbf{x} \neq 0$, $\mathbf{x}^*\mathbf{Ax} > 0$, por otro lado si $\mathbf{x}^*\mathbf{Ax} \geq 0$, entonces **A** es una matriz semidefinida.

La condición necesaria y suficiente para que **A** sea una matriz positiva definida es que se cumpla alguna de las siguientes condiciones:

1.- Todos los menores principales de la diagonal son positivos.
2.- Todos los valores característicos de **A** sean positivos.
3.- **A** pueda representarse en la forma $\mathbf{A} = \mathbf{QQ}^*$, en donde **Q** es una matriz no singular.

# APÉNDICE II

## CÁLCULO DEL GRADIENTE DEL MSE

De la ecuación (3.25) que representa el MSE en forma matricial, se tiene:

$$\xi = \sigma_d^2 + \mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p}$$

El cálculo del gradiente de MSE es,

$$\begin{aligned}
\nabla_{MSE} &= \frac{\partial \xi}{\partial \mathbf{h}} \\
&= \frac{\partial\left[\sigma_d^2 + \mathbf{h}^t\mathbf{R}\mathbf{h} - 2\mathbf{h}^t\mathbf{p}\right]}{\partial \mathbf{h}} \\
&= \frac{\partial\sigma_d^2}{\partial \mathbf{h}} + \frac{\mathbf{h}^t\mathbf{R}\mathbf{h}}{\partial \mathbf{h}} - \frac{2\mathbf{h}^t\mathbf{p}}{\partial \mathbf{h}} \\
&= \frac{\partial\sigma_d^2}{\partial \mathbf{h}} + \frac{\mathbf{R}\mathbf{h}^2}{\partial \mathbf{h}} - \frac{2\mathbf{h}^t\mathbf{p}}{\partial \mathbf{h}} \\
&= 0 + 2\mathbf{R}\mathbf{h} - 2\mathbf{p} \\
&= 2\mathbf{R}\mathbf{h} - 2\mathbf{p}
\end{aligned}$$

# APÉNDICE III

## Programa ANC.C para el DSP

```
/*
 *
 * ANC.C - Adaptive Noise Canceller for the TMS320C30
 *
 * Atzin C  1997
 *
 */

#include <stdlib.h>
#include <math.h>

#include "c30lsi.h"
#include "anc.h"

/*
 * Prototype of DSP interrupt handler
 */
void c_int01 ();

/*
 * Message handlers
 */

void dsp_reset (void);
void dsp_init (void);
void dsp_start (void);
void dsp_stop (void);
void dsp_set_id(void);
void dsp_unset_id(void);

typedef void (*HANDLER) ();
HANDLER handle [DSP_LAST] =
{
 NULL,
 &dsp_reset,
 &dsp_init,
 &dsp_start,
 &dsp_stop,
 &dsp_set_id,
 &dsp_unset_id
};

/*
 * Global variables
 */

int n = 0;
int id = 0;
int Sh = Shifter;
int Fl = Filter_length;
int ref = REF_OFF;
int state = DSP_OFF;
int delay = DELAY;
float (*H) = NULL;
float (*X) = NULL;
float mu = MU;
float rate = SAMPLE_RATE_HZ;
volatile int dspMessage;

void dsp_reset ()
{
 mu = MU;
 delay = DELAY;
 rate = SAMPLE_RATE_HZ;
 C30_SET_INTERRUPT(1,c_int01);
 C30_SET_TIMER1(rate);
 C30_ADC_START();
 state = DSP_ON;
}

/*
 * Service routines
 */

void dsp_init ()
{
 n = 0;
 if (H != NULL) free (H);
 H = calloc (Fl, sizeof (float));
 if (X != NULL) free (X);
 X = calloc (Sh, sizeof (float));
}

void dsp_start ()
{
 C30_SET_INTERRUPT(1,c_int01);
 C30_SET_TIMER1(rate);
 C30_ADC_START();
 state = DSP_ON;
}

void dsp_stop ()
{
 C30_ADC_STOP ();
 C30_SET_TIMER1 (0);
 free (H);
 free (X);
 state = DSP_OFF;
}

void dsp_set_id ()
{
 id = 1;
}

void dsp_unset_id ()
{
 id = 0;
}

void main (void)
{
 for (;;)
 {
  dspMessage = DSP_READY;
  while (dspMessage == DSP_READY);
  handle [dspMessage] ();
 }
}
```

```
/*
 * Interrupt service routine
 */

void  c_int01()
{
 float Primary_input;
 float y;
 float e, e_mu;
 float *px;
 float *ph;
 int A1, A2;
 int k,l;

/*
 * Input Signal Adquisition
 */

 Primary_input = ADC(0);
 if (ref == REF_OFF)
 {
  A1 = Sh;
  A2 = A1 - Fl;
  n = (n - 1) & (A1 - 1);
  X[n] = Primary_input;
  l = (n - delay) & (A1 - 1);
 }
 else
 {
  A1 = Fl;
  A2 = 0;
  n = (n - 1) & (A1 - 1);
  X[n] = ADC(1);
  l = n;
 }

 /*
  * Convolution
  */

 y = 0.0;
 if (l >= A2)
 {
  k = A1 - l;
  px = X + l;
  ph = H;
  while (k--) y += *px++ * *ph++;
  k = l - A2;
  px = X;
  while (k--) y += *px++ * *ph++;
 }
 else
 {
  px = X + l;
  ph = H;
  k = Fl;
  while (k--) y += *px++ * *ph++;
 }

 /*
  * Adapting routine
  */

 e = Primary_input - y;
 e_mu = e * mu;
 if (l >= A2)
 {
  k = A1 - l;
  px = X + l;
  ph = H;
  while (k--) *(ph++) += e_mu * *px++;
  k = l - A2;
  px = X;
  while (k--) *(ph++) += e_mu * *px++;
 }
 else
 {
  px = X + l;
  ph = H;
  k = Fl;
  while (k--) *(ph++) += e_mu * *px++;
 }

 /*
  * Output Signals
  */

  DAC(0) = e;
  DAC(1) = y;
}
```

# APÉNDICE IV
## Programa ANCPC.C para la PC

```
/*
 * ANCPC.C - PC driver for program ANC.OUT running into
             TMS320C30
 * Atzin C. 1997
 */


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <malloc.h>


#include "c30coff.h"
#include "tms30.h"
#include "anc.h"
#include "..\getopt\getopt.h"
#include "..\dspanc\dspanc.h"


#define OK                      0
#define ERROR                   1
#define C30_ADRESS              0x0390
#define C30_FILE                "anc.out"
#define     VERSION             "This is ANCPC program
                                Version 1.0 PC Handler"


static char version[] = VERSION;

static char usage[] = VERSION "\n"
"Usage:\n"
"           ancpc [-f f_length -r rate -m mu -d delay -n -v]\n"
"           ancpc -l\n"
"           ancpc -s\n"
"           ancpc -e\n"
"           ancpc -t\n"
"           ancpc -h\n\n"
" -f f_length : filter length, default 32\n"
" -r rate     : sampling rate, default 10kHz\n"
" -m mu       : adaptive constant, default 1e-20\n"
" -d delay    : delay samples, default 0\n"
" -n : reference option, receive reference input\n"
" -l : load DSP program to the board\n"
" -s : stop execution\n"
" -e : reset the board\n"
" -t : dsp program test\n"
" -v : verbose option, print details during execution\n"
" -h : help and exit\n";

static char bad_usage[] = "Error in command line  Try: ancpc -
                          h";

static char optlist[] = "f:r:m:d:nlsetvh";

/*          Global variables
 */

int verbose = 0;

long id;
long dsp_state;
long f = Filter_length;
long delay = DELAY;
long ref = REF_OFF;
double mu = MU;
double rate = SAMPLE_RATE_HZ;
double delaysec = DELAY;

/*          DSP routines
 */

int dsp_load(void)
{
 int result;

 /*          Load DSP program
 */

 if (dspLoad(C30_FILE)) return(1);

 /*          Load DSP Symbols
 */

 if (c30LoadSymbols(C30_FILE)) return(2);

 /*          Test communication with DSP
 */

 result = dspSendMessage(DSP_READY);
 if (result == 1) return(1);
 if (result == 2) return(2);

 /*          Release DSP Symbols
 */

 c30ReleaseSymbols();

 return OK;
}

/*          Target Reset
*/

int dsp_reset(void)
{
 if ((SelectBoard(C30_ADRESS)) != C30_ADRESS) return
(1);
 Reset();
 c30ReleaseSymbols();
 remove("outdata");

 return OK;
}

/*          ID Test
*/

int dsp_test_id(void)
{
 clock_t t;
```

```
dspSendMessage(DSP_SET_ID);
dspSendMessage(DSP_READY);
t = clock() + 50;
while (clock() < t);
if ((dspGet1Long("id", NULL)) != 1) goto fail_id;
dspSendMessage(DSP_UNSET_ID);
dspSendMessage(DSP_READY);
t = clock() + 50;
while (clock() < t);
if ((dspGet1Long("id", NULL)) != 0) goto fail_id;

return OK;

fail_id : return(1);
}

/*          DSP Test
*/

int dsp_test(void)
{
clock_t t;
FILE *fp;

/*          Select target without affecting any program running
on it
*/

if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return
(1);

/*          Load DSP Symbols
*/

if (c30LoadSymbols(C30_FILE)) return(2);

/*          ID Test
*/

if (dsp_test_id() != OK) return(3);
fp = fopen("outdata", "w");
fprintf(fp, "%i\n", dspGet1Long("FI", NULL));
fprintf(fp, "%g\n", dspGet1Float("rate", NULL));
fprintf(fp, "%g\n", dspGet1Float("mu", NULL));
fprintf(fp, "%i\n", dspGet1Long("delay", NULL));
delaysec = dspGet1Long("delay", NULL) *
(1 / dspGet1Float("rate", NULL));
fprintf(fp, "%g\n", delaysec);
fprintf(fp, "%i\n", dspGet1Long("ref", NULL));
fprintf(fp, "%i\n", dspGet1Long("state", NULL));
fclose (fp);
t = clock() + 500;
while (clock() < t);

/*          Release DSP Symbols
*/

c30ReleaseSymbols();

return OK;
}

/*          Run DSP Program
*/

int dsp_run(void)
{

/*          Select target without affecting any program running
on it
*/

if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return
(1);

/*          Load DSP Symbols
*/

if (c30LoadSymbols(C30_FILE)) return(2);

/*          ID Test
*/

if (dsp_test_id() != OK) return(3);
dspSendMessage(DSP_STOP);
dspSendMessage(DSP_READY);
if (verbose) fprintf(stdout, "f = %i\n", f);
if (dspSet1Long("FI", f)) goto fail;
if (verbose) fprintf(stdout, "rate = %gkHz\n", rate/1000.0);
if (dspSet1Float("rate", (float) rate)) goto fail;
if (verbose) fprintf(stdout, "mu = %g\n", mu);
if (dspSet1Float("mu", (float) mu)) goto fail;
if (verbose) fprintf(stdout, "delay = %i\n", delay);
if (dspSet1Long("delay", delay )) goto fail;
delaysec = (float) delay * (1 / rate);
if (verbose) fprintf(stdout, "delaysec = %g\n", delaysec);
if (verbose) fprintf(stdout, "reference = %i\n", ref);
if (dspSet1Long("ref", ref)) goto fail;

dspSendMessage(DSP_INIT);
dspSendMessage(DSP_START);
dspSendMessage(DSP_READY);

/*          Release DSP Symbols
*/

c30ReleaseSymbols();

return OK;

fail: return (4);
}

/*          Stop DSP Program
*/

int dsp_stop(void)
{

/*          Select target without affecting any program running
on it
*/

if ((WarmSelect(C30_ADRESS)) != C30_ADRESS) return
(1);

/*          Load DSP Symbols
*/

if (c30LoadSymbols(C30_FILE)) return(2);

/*          ID Test
*/

if (dsp_test_id() != OK) return(3);
```

```
/*          Get DSP program state
*/

if ((dspGetILong("state", NULL)) != DSP_ON) return(4);

dspSendMessage(DSP_STOP);
dspSendMessage(DSP_READY);

/*  Release DSP Symbols
*/

c30ReleaseSymbols();

return OK;
}

/*          PC main function
*/

int main(int argc, char *argv[])
{
 int opt;

 while ((opt = getopt(argc, argv, optlist)) != EOF)
  {
switch (opt)
 {
 case 'f': f = atoi(optarg);break;
 case 'r': rate = atof(optarg); break;
 case 'm': mu = atof(optarg); break;
 case 'd': delay = atoi(optarg); break;
 case 'n': ref = REF_ON;break;
 case 'l' fprintf(stdout,"%i\n",dsp_load());
 exit(EXIT_SUCCESS);
 case 's': fprintf(stdout, "%i\n",dsp_stop());
exit(EXIT_SUCCESS);
 case 'e': fprintf(stdout, "%i\n",dsp_reset());
 exit(EXIT_SUCCESS);
 case 't': fprintf(stdout, "%i\n",dsp_test());
 exit(EXIT_SUCCESS);
 case 'v': verbose = 1 ; break;
 case 'h': puts(usage); exit(EXIT_SUCCESS);
 default: puts(bad_usage); exit(EXIT_FAILURE);
 }
}
fprintf(stdout,"%i\n",dsp_run());

return OK;
}
```

# APÉNDICE V
## Encabezado C30LSI.H

```
/*
 * C30LSI.H - Utilities to write DSP applications in C for the
              TMS320C30
 * (Hardware especific to the LSI C30 System Card )
 *
 * Target: TMS320-C30 / LSI
 *
 * (C) Felipe Orduna 1992
 */


/* Pointers to hardware registers
 */
#define C30_PBUS_CTRL       (*(long *)0x808064)
#define C30_XBUS_CTRL       (*(long *)0x808060)
#define C30_TIMER1_CTRL     (*(long *)0x808030)
#define C30_TIMER1_COUNT (*(long *)0x808038)


/* Macros for AD and DA conversion
 */
#define C30_ADC_UNITS_PER_VOLT    7.158278826e+8
#define C30_ADC_BASE                  0x804000
#define ADC(n)                    (long)(*(long *)
                                  (C30_ADC_BASE + (n)))
#define DAC(n)                    (*(long *)
                                  (C30_ADC_BASE + (n)))

/* Calibration factors for AD and DA conversion
 *
 *    ADC = (2^(31) / 3) [units per Volt]
 *    DAC = (2^(31) / 3) [units per Volt]
 *
 */
#define C30_ADC_UNITS_PER_VOLT  7.158278826e+8
#define C30_DAC_UNITS_PER_VOLT  7.158278826e+8
#define C30_ADC_VOLTS_PER_UNIT  1.396983862e-9
#define C30_DAC_VOLTS_PER_UNIT  1.396983862e-9


/* Convert sampling frequency in Hz to timer count value
 */
#define C30_GET_COUNT(fs)  (unsigned long)
                           (8.3333333e+6/(float)(fs))

/* Set Timer #1 to especified sampling frequency in H
 */
#define C30_SET_TIMER1(fs)                          \
{                                                   \
  C30_PBUS_CTRL = 0x0800,                           \
  C30_XBUS_CTRL = 0x0000,
                \
  C30_TIMER1_CTRL = 0x0601,                         \
  C30_TIMER1_COUNT = C30_GET_COUNT(fs);             \
  C30_TIMER1_CTRL = 0x06C1,                         \
}

/* Prepare calls on interrupt n to the function at the especified
address
 */
#define C30_SET_INTERRUPT(n,adrs)                   \
{                                                   \
  asm("   AND    00001FFFh,ST"),                    \
  (*(unsigned long *)(n + 1)) = (unsigned long)adrs, \
}

/* Enable interrupts (from the analog to digital converter)
 */
#define C30_ADC_START()                             \
{                                                   \
  asm("   OR    00000002h,IE"),
                \
  asm("   OR    00002000h,ST"),                     \
}


/* Disable interrupts (from the analog to digital converter)
 */
#define C30_ADC_STOP()                              \
{                                                   \
  asm("   AND    0FFFFFFFDh,IE"),                   \
  asm("   AND    0FFFFDFFFh,ST"),                   \
}

/* This bit of code declares a global label to the C entry point
"_c_int00"
 * (defined in RTS30.LIB) where the processor is initialized
and
 * the user's main() function is called.
 */
asm("   .sect  \".init\""),
asm("   .word  _c_int00"),
asm("   .globl _c_int00"),
```

# APÉNDICE VI
## Encabezado ANC.H

```
/*
 * ANC.H - This is a "header" for using with the ANC C & ANCPC.C
 */


/*Define default settings
*/

#define   SAMPLE_RATE_HZ          10000.0      // Sampling rate in Hz
#define   Filter_length           32           // Filter length
#define   Shifter                 256          // Shifter size
#define   MU                      1e-20        // Adaptive constant
#define   DELAY                   0            // Delay samples


/* Constants for DSP - PC message protocol
*/

#define   DSP_READY               0
#define   DSP_RESET               1
#define   DSP_INIT                2
#define   DSP_START               3
#define   DSP_STOP                4
#define   DSP_SET_ID              5
#define   DSP_UNSET_ID            6
#define   DSP_LAST                7

/* Constant for DSP state
*/

#define   DSP_OFF           0
#define   DSP_ON            1

/* Constant for Reference Input
*/

#define   REF_OFF           0
#define   REF_ON            1
```

# APÉNDICE VII
## Encabezado DSPANC.H

```
/*
 * DSPANC.H - Funtion prototypes for DSP - PC control
 */

int dspLoad( char *fname );

long dspGet1Long( char *name, int *rcode );

float dspGet1Float( char *name, int *rcode );

int dspSet1Long( char *name, long value );

float dspSet1Float( char *name, float value );

int dspSendMessage( unsigned long msg );
```

# APÉNDICE VIII
## Encabezado C30COFF.H

```
/*
 * C30COFF.H - Pointer retrieval of global symbols in
                        TMS320C30 COFF files
 *
 * Target: i80x86
 *
 * (C) Felipe Orduna 1992 (Based on MTT's C30IF library)
 */


typedef     struct
{
   unsigned short  magic;
   unsigned short  nsect;
   unsigned long   stamp;
   unsigned long   psyms;
   unsigned long   nsyms;
   unsigned short  nprop;
   unsigned short  flags;
} C30_COFF_HEADER;


typedef     struct
{
   union
   {
          char symbol[8];
          struct
          {
             unsigned long here;
             unsigned long symptr;
          } symp;
   } sym;
   long         symval;
   unsigned short  symsect;
   unsigned short  symtype;
   unsigned char   symclass;
   unsigned char   symaux;
} C30_COFF_SYMBOL;


typedef     struct c30_list
{
   char         *name;
   unsigned long   value;
   struct c30_list *next;
} C30_LIST;


#define     C_NULL              0
#define     C_AUTO              1
#define     C_EXT               2
#define     C_STAT              3
#define     C_EXTDEF            4
#define     C_LABEL             5

#define     TMS320C30           0x93

#define     C30_MAXSYMLEN 80


/* Global pointer to list of C30 symbols
 */
extern C30_LIST *_c30list;


/* Function prototypes
 */
int c30LoadSymbols( char *fname );
int c30ReleaseSymbols( void );
unsigned long c30GetPointer( char *name );
```

# APÉNDICE IX
## Programa DSPANC.C

```
/*
 * DSPANC.C - Program for interfacing DSP with PC driver
 *
 * Based on 'C30 SPECTRUM's Library
 */

#include <time.h>

#include "tms30.h"
#include "c30coff.h"
#include "anc.h"

#define C30_ADRESS          0x0390    // 'C30 Address
#define DSP_MSG_NAME        "dspMessage"// DSP Message
#define LAPSE               1         // Waiting time for
                                      a job

unsigned long dspMessage = DSP_READY;


/*      Load DSP program to TMS320C30 board at 'C30
Address
*/

int dspLoad( char *fname )
{
        if( SelectBoard(C30_ADRESS) != C30_ADRESS )
                return(1);

        if( LoadObjectFile(fname) )
                return(2);

        Reset();

        return(0);
}

/*      Reads a 32 bit integer from 'C30 memory at the
given address
*/

long dspGetLong( char *name, int *rcode )
{
        unsigned long ptr;
        long value;

        if( ( ptr = c30GetPointer( name ) ) == 0 )
        {
                if( rcode != NULL )
                        *rcode = 1;
                return(0);
        }
        value = GetInt( ptr, DUAL );

        if( rcode != NULL )
                *rcode = 0;
                return(value);
}

/*      Takes a 32 bit 'C30 format floating point value from
the
*       board at the address specified
*/

float dspGetFloat( char *name, int *rcode )
{
        unsigned long ptr;
        float value;

        if( ( ptr = c30GetPointer( name ) ) == 0 )
        {
                if( rcode != NULL )
                        *rcode = 1;
                return(0);
        }
        RdBlkFlt( ptr, DUAL, 1, &value);

        if( rcode != NULL )
                *rcode = 0;

        return(value);
}

/*      Puts a 32 bit integer down to the indicated memory
location on 'C30
*/

int dspSetLong( char *name, long value )
{
        unsigned long ptr;

        if( ( ptr = c30GetPointer( name ) ) == 0 )
                return(1);

        if( PutInt( ptr, DUAL, value ) )
                return(2);

        return(0);
}

/*      Deposites a floating point value into board memory
at the given address
*/
int dspSetFloat( char *name, float value )
{
        unsigned long ptr;

        if( ( ptr = c30GetPointer( name ) ) == 0 )
                return(1);

        if( PutFloat( ptr, DUAL, value ) )
                return(2);

        return(0);
}

/*      Sends a Message for invoking a routine on the 'C30
board
*/

int dspSendMessage( unsigned long msg )
{
        time_t time_out;
        time_t lapse = LAPSE;
```

```
/* Wait until C30 puts dspMessage = DSP_READY
 */
        time_out = time(NULL) + lapse;
        do
        {
                dspMessage = dspGetLong
                ( DSP_MSG_NAME, NULL );
                if( time(NULL) > time_out )
                        return(1);
        }
        while( dspMessage != DSP_READY );

/* Sends the message
 */
        dspMessage = msg;
        if( dspSetLong( DSP_MSG_NAME, dspMessage )
)
                return(2);

        return(0);
}
```

# APÉNDICE X
## Programa ANC.CPP

```
/
// anc.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"

#include "anc.h"
#include "dialog.h"

/////////////////////////////////////////////////////////////////////////////
// CMainWindow

CTheApp NEAR theApp;

CMainWindow::CMainWindow()
{
        RECT    InitPos;

        InitPos.left = 0;
        InitPos.top = 0;
        InitPos.right = 640;
        InitPos.bottom = 400;

        Create( NULL, "ANC FILTER",
        WS_OVERLAPPEDWINDOW, InitPos , NULL,
        "AncMenu" );
        LoadAccelTable("ANCAccelTable");
}

BEGIN_MESSAGE_MAP(CMainWindow, CFrameWnd)
        //{{AFX_MSG_MAP(CMainWindow)
        ON_COMMAND(ID_HELP_HELP, OnHelpHelp)
        ON_WM_INITMENU()
        ON_COMMAND(ID_EXECUTE_EXECUTE,
        OnExecuteExecute)
        ON_COMMAND(ID_EXECUTE_EXIT,
        OnExecuteExit)
        ON_COMMAND(ID_HELP_ABOUTANC,
        OnHelpAboutanc)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


BOOL CTheApp::InitInstance()
{

        SetDialogBkColor();    // hook gray dialogs (was
                               default in MFC V1)
        m_pMainWnd = new CMainWindow();
        m_pMainWnd->ShowWindow( m_nCmdShow );
        m_pMainWnd->UpdateWindow();

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainWindow message handlers


void CMainWindow::OnHelpHelp()
{
        CString Help_line;

        Help_line = "WinHelp help.hlp";
        if (WinExec(Help_line, SW_SHOW) < 32)
        AfxMessageBox("Can't execute command");
}


void CMainWindow::OnExecuteExecute()
{
        CMainDialog dlg;
        dlg.DoModal();
}

void CMainWindow::OnExecuteExit()
{
        if (AfxMessageBox("Are you sure you want to
        leave ?",    MB_YESNO|
        MB_ICONINFORMATION) == IDYES)
        exit (0);
}

void CMainWindow::OnHelpAboutanc()
{
        CDialog About( "AboutBox", this );
        About.DoModal();
}
```

# APÉNDICE XI
## Programa DIALOG.CPP

```
// dialog.cpp : implementation file
//

#include <string.h>

#include "stdafx.h"
#include "resource.h"
#include "dialog.h"

// Define Error Messages
//

#define E000        "Error : Can't Execute Command",
#define E001        "Error : Can't find target",
#define E002        "Error : Can't load C30 Symbols",
#define E003        "Error : Lost communication with DSP
                    program",
#define E004        "Error : DSP program is not running";

#define   P_ProgName          "ancpc"
#define   GETHINST(hWnd)   ((HINSTANCE)
          GetWindowWord(hWnd, GWW_HINSTANCE))

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

CString EMessage;
CString DelaySec;

/////////////////////////////////////////////////////////////////////////
// CMainDialog dialog

CMainDialog::CMainDialog()
          CDialog(CMainDialog::IDD)
{
          //{{AFX_DATA_INIT(CMainDialog)
          m_FilterLength = "32";
          m_Delay = "0";
          m_AdaptiveCte = "1e-020";
          m_SampleRate = "10000";
          m_ReferenceInput = FALSE;
          m_DelaySec = "0";
          //}}AFX_DATA_INIT
}


void CMainDialog::DoDataExchange(CDataExchange* pDX)
{
          CDialog::DoDataExchange(pDX);
          //{{AFX_DATA_MAP(CMainDialog)
          DDX_Control(pDX, IDC_STOP, m_Stop);
          DDX_Control(pDX, IDC_LOAD, m_Load);
          DDX_Control(pDX, IDC_EXECUTE, m_Execute);
          DDX_Control(pDX, IDC_REFRESH, m_Refresh);
          DDX_Text(pDX, IDC_FILTERLENGTH,
                    m_FilterLength);
          DDX_Text(pDX, IDC_DELAY, m_Delay);
          DDX_Text(pDX, IDC_ADAPTIVECTE,
                    m_AdaptiveCte);
          DDX_Text(pDX, IDC_SAMPLERATE,
                    m_SampleRate);
          DDX_Check(pDX, IDC_REFERENCE,
                    m_ReferenceInput);
          DDX_Text(pDX, IDC_DELAYSEC, m_DelaySec);
          //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMainDialog, CDialog)
          //{{AFX_MSG_MAP(CMainDialog)
          ON_BN_CLICKED(IDC_EXECUTE,
                    OnClickedExecute)
          ON_BN_CLICKED(IDC_STOP, OnClickedStop)
          ON_BN_CLICKED(IDC_LOAD, OnClickedLoad)
          ON_BN_CLICKED(IDC_REFRESH,
                    OnClickedRefresh)
          //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// Routine for the WinExec & Wait
//

typedef struct _Process_Info
{
 HINSTANCE hInstance;
 HWND hWnd;
} Process_Info;

BOOL CALLBACK EnumWndProc(HWND hWnd, LPARAM
lParam)
{
 Process_Info *pi = (Process_Info *) lParam;

 if (GETHINST(hWnd) == pi->hInstance)
 {
  pi->hWnd = hWnd;
  return FALSE;
 }
 return TRUE;
}

UINT Win_Exec(HWND hWnd, CString command, int show,
int wait)
{
 UINT result;
 Process_Info pi;
 WNDENUMPROC enumproc;
 HCURSOR hcur, hcur_old;
 MSG msg;

 result = WinExec(command, show);
 if (result < 32) return result;
 if (!wait) return result;

 pi.hInstance = (HINSTANCE) result;
 pi.hWnd = NULL;

 enumproc = (WNDENUMPROC)
MakeProcInstance((FARPROC)EnumWndProc,
GETHINST(hWnd));
 EnumWindows(enumproc, (LPARAM) &pi);
 FreeProcInstance((FARPROC)enumproc);

 hcur = LoadCursor(NULL, IDC_WAIT);
```

```
 hcur_old = SetCursor(hcur);
 for (;;)
 {
  if (!IsWindow(pi.hWnd)) break;
  if (GETHINST(pi.hWnd) != pi.hInstance) break;
  if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    SetCursor(hcur);
  }
 }
 SetCursor(hcur_old);
 return result;
}

UINT ExecuteCommand(HWND HWnd, CString Command)
{
        FILE    *anchat, *result;
        CString rslt;

        Command += CString(" > result");
        anchat = fopen("anc.bat","w");
        fprintf(anchat, "%s", Command);
        fclose(anchat);
        if (Win_Exec(HWnd, "anc.bat", SW_HIDE, 1) <
32)
        {
                EMessage = E000;
                return (1);
        }
        result = fopen("result", "r");
        fscanf(result, "%s", rslt);
        fclose(result);
        remove("result");
        remove("anc.bat");
        if (rslt != "0")
        {
                if (rslt == "1") EMessage = E001;
                if (rslt == "2") EMessage = E002;
                if (rslt == "3") EMessage = E003;
                if (rslt == "4")
                {
                        EMessage = E004;
                        return (4);
                }
                return (1);
        }
        return (0);
}

/////////////////////////////////////////////////////////////////////////////
// CMainDialog message handlers

BOOL CMainDialog::OnInitDialog()
{
        CDialog::OnInitDialog();

        CString Test, ref, state;
        FILE    *test;

        Test = CString(P_ProgName) + " -t";
        if (ExecuteCommand(m_hWnd, Test) == 0)
        {
                test = fopen("outdata","r");
                fscanf(test, "%s", m_FilterLength);
                fscanf(test, "%s", m_SampleRate);
                fscanf(test, "%s", m_AdaptiveCte);
                fscanf(test, "%s", m_Delay);
                fscanf(test, "%s", m_DelaySec);
                fscanf(test, "%s", ref);
                m_ReferenceInput = FALSE;
                if (ref == "1")
                        m_ReferenceInput = TRUE;
                fscanf(test, "%s", state);
                m_Stop.EnableWindow(TRUE);
                if (state == "0")
                m_Stop.EnableWindow(FALSE);
                fclose(test);
                remove("outdata");
                UpdateData(FALSE);
                m_Execute.EnableWindow(TRUE);
                m_Refresh.EnableWindow(TRUE);
                m_Load.SetWindowText("Unload");
        }

        CenterWindow();

        return TRUE;
}

void CMainDialog::OnClickedLoad()
{
        CString load_dsp_program;
        CString load_state;
        m_Load.GetWindowText(load_state);
        if ( load_state == "Load")
        {
                load_dsp_program =
                        CString(P_ProgName) + " -l";
                if (ExecuteCommand(m_hWnd,
                        load_dsp_program) == 0)
                {
                m_Execute.EnableWindow(TRUE);
                m_Stop.EnableWindow(FALSE);
                m_Refresh.EnableWindow(TRUE);
                m_Load.SetWindowText("Unload");
                }
                else
                        {
                        AfxMessageBox(EMessage);
                        EndDialog(0);
                        }
        }
        else
        {
                load_dsp_program =
                        CString(P_ProgName) + " -
e";
                if (ExecuteCommand(m_hWnd,
                        load_dsp_program) == 0)
                {
                m_Execute.EnableWindow(FALSE);
                m_Stop.EnableWindow(FALSE);
                m_Refresh.EnableWindow(FALSE);
                m_Load.SetWindowText("Load");
                }
                else
                        {
                        AfxMessageBox(EMessage);
                        EndDialog(0);
                        }
        }
}

void CMainDialog::OnClickedExecute()
```

```
{
        CString command_line;

        UpdateData();
        command_line = CString(P_ProgName) + (" -r " +
                m_SampleRate);
        command_line += (CString(" -f ") +
m_FilterLength);
        command_line += (CString(" -m ") +
                m_AdaptiveCte);
        command_line += (CString(" -d ") + m_Delay);
        if (m_ReferenceInput == TRUE)
                command_line += (CString(" -n "));
        if (ExecuteCommand(m_hWnd, command_line) =
0)

        {
                OnClickedRefresh();
                m_Stop.EnableWindow(TRUE);
        }
        else
                {
                        AfxMessageBox(EMessage);
                        EndDialog(0);
                }
}

void CMainDialog::OnClickedStop()
{
        CString stop_execution;
        UINT    StopResult;

        stop_execution = CString(P_ProgName) + " -s";
        StopResult = ExecuteCommand(m_hWnd,
                stop_execution);
        if ( StopResult == 0)
                m_Stop.EnableWindow(FALSE);
        else
                {
                AfxMessageBox(EMessage);
                if (StopResult == 4)
                m_Stop.EnableWindow(FALSE);
                else
                        EndDialog(0);
                }
}

void CMainDialog::OnClickedRefresh()
{
        CString Refresh;
        CString ref;
        FILE    *rfsh;

        Refresh = CString(P_ProgName) + " -t";
        if (ExecuteCommand(m_hWnd, Refresh) == 0)
        {
                rfsh = fopen("outdata","r");
                fscanf(rfsh, "%s", m_FilterLength);
                fscanf(rfsh, "%s", m_SampleRate);
                fscanf(rfsh, "%s", m_AdaptiveCte);
                fscanf(rfsh, "%s", m_Delay);
                fscanf(rfsh, "%s", m_DelaySec);
                fscanf(rfsh, "%s", ref);
                m_ReferenceInput = FALSE;
                if (ref == "1")
                        m_ReferenceInput = TRUE;
                fclose(rfsh);
                remove("outdata");
                UpdateData(FALSE);
        }
        else
                {
                        AfxMessageBox(EMessage);
                        EndDialog(0);
                }
}
```

# APÉNDICE XII

## Encabezado ANC.H para el programa para Windows©

```
// anc.h : header file
//

#ifndef _ANC_H_
#define _ANC_H_
/////////////////////////////////////////////////////////////////////////////
// CMainWindow frame

class CMainWindow : public CFrameWnd
{
public:
   CMainWindow();

            // Generated message map functions
            //{{AFX_MSG(CMainWindow)
            afx_msg void OnHelpHelp();
            afx_msg void OnExecuteExecute();
            afx_msg void OnExecuteExit();
            afx_msg void OnHelpAboutanc();
            //}}AFX_MSG
            DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
class CTheApp : public CWinApp
{
public:
            BOOL InitInstance();
};

#endif //_ANC_H_
```

# APÉNDICE XIII

## Encabezado DIALOG.H para el programa en Windows©

```
// dialog.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CMainDialog dialog

class CMainDialog : public CDialog
{
// Construction
public
        CMainDialog();

// Dialog Data
        //{{AFX_DATA(CMainDialog)
        enum { IDD = ANCDIALOG };
        CButton   m_Stop;
        CButton   m_Load;
        CButton   m_Execute;
        CButton   m_Refresh;
        CString   m_FilterLength;
        CString   m_Delay;
        CString   m_AdaptiveCte;
        CString   m_SampleRate;
        BOOL      m_ReferenceInput;
        CString   m_DelaySec;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange*
                pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CMainDialog)
        afx_msg void OnClickedExecute();
        afx_msg void OnClickedStop();
        afx_msg void OnClickedLoad();
        virtual BOOL OnInitDialog();
        afx_msg void OnClickedRefresh();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

# REFERENCIAS Y BIBLIOGRAFÍA

## REFERENCIAS


[Ref. 1] Kernighan Brian W. & Ritchie Dennis M., *El lenguaj de prgramación C*, Prentice-Hall, México, 1985.

[Ref. 2] Mitra Sanjit K. & Kaiser James F., *Handbook for Digital Signal Processing*, Wiley-Interscience publication, U.S.A., 1993.

[Ref. 3] Peter M Clarkson, *Optimal and Adaptive Signal Processing*, CRC Press, U.S.A., 1993.

[Ref. 4] Proakis John G *et all*, *Advanced Digital Signal Processing*, Macmillan Publishing Company, New York, 1992.

[Ref. 5] Rabiner Lawrence & Bernand Gold, *Theory and Application of Digital Signal Processing*, Prentice Hall, USA, 1975.

[Ref. 6] Stearns Samuel D. & Hush Don R., *Digital Signal Analysis*, Prentice-Hall Inc., U.S.A., 1990

[Ref. 7] Texas Instruments, *TMS320 Floating-Point DSP Assembly Language Tools.*

[Ref. 8] Texas Instruments, *TMS320 Floating-Point DSP Optimizing C Compiler.*

[Ref. 9] Widrow B. & Stearns S.D., *Adaptive Signal Processing*, Prentice-Hall Inc., U.S.A, 1985.


## BIBLIOGRAFÍA


Acroyd Martin H., *Digital Filters*, Buterworth & Co., Great Britain, 1973.

Chassaing Rulph, *Digital signal Processing with C and the TMS320C30*, John wiley & Sons Inc., USA, 1992.

Clarkson Peter M., *Optimal and Adaptive Signal Processing*, CRC Press, U.S., 1993.

Collazo, Javier L. , *Diccionario Enciclopédico de Términos Técnicos, Inglés-Español, Español-Inglés*, McGrawHill, U.S.A., 1980.

Deif Assem S., *Advanced matrix theory for scientists and engineers*, Abacus press - Halsted press, Great Britain 1982.

Lapedes Daniel N., *Dictionary Scientific and Technical Terms*, McGrawHill, 2a. ed., Philipines, 1976.

Haykin Simon S., *Adaptive Filter Theory*, Prentice-Hall, 2a. ed., U.S.A.,1991.

Mitra Sanjit K. & Kaiser James F., *Handbook for Digital Signal Processing*, Wiley-Interscience publication, U.S.A. 1993.

Morrison R., *Noise and other interfering signals*, Wiley-Interscience publication, U.S.A., 1992.

Proakis John G., *et all, Advanced Digital Signal Processing*, Macmillan Publishing Company, New York, 1992.

Proakis John G. & Manolakis Dimitris G., *Digital Signal Processing: Principles, Algorithms & Applications*, Prentice Hall, U.S.A., 1996.

Rabiner Lawrence R. & Bernard Gold, *Theory and Application of Digital Signal Processing*, Prentice Hall, USA, 1975.

Stearns Samuel D. & Hush Don R., *Digital Signal Analysis*, Prentice-Hall Inc., U.S.A., 1990.

Widrow B. & Stearns S.D., *Adaptive Signal Processing*, Prentice-Hall, Inc., U.S.A, 1985.

## ARTÍCULOS

Brown Allen, *Digital filters adapt, Electronics World + Wireless World*, March 1995.

Hutchings Howard, *Audio processing on the PC*, Electronics World, November 1996.

Lawson Stuart, *Theory & design of digital filters*, Electronic Engineering, May 1977.

Lee Edward A., *Programmable DSP Architectures*, IEEE ASSP Magazine, October 1988.

Widrow B. et all, *Adaptive Noise Cancelling: Principles & Applications*, Proceedings of the IEEE, Vol. 63, No. 12, December 1975.